UNIVERSIDAD CARLOS III DE MADRID

Escuela Politécnica Superior

Ingeniería Técnica de Telecomunicación: Telemática

PROYECTO FIN DE CARRERA

Diseño e implementación de una tienda electrónica mediante STRUTS y SOAP

Rocío López Valladolid

2009

UNIVERSIDAD CARLOS III DE MADRID

Escuela Politécnica Superior

Ingeniería Técnica de Telecomunicación: Telemática

PROYECTO FIN DE CARRERA

Diseño e implementación de una tienda electrónica mediante STRUTS y SOAP

Autor: Rocío López Valladolid

Director: Dr. Abelardo Pardo Sánchez

TRIBUNAL:

Presidente: Vicente Luque Centeno

Secretario: Marcelo G. Bagnulo Braun

Vocal: Fausto J. Sainz Salces

CALIFICACIÓN:.....

FECHA:.....

Resumen.

Gracias al uso de Internet como medio de comunicación, las aplicaciones Web han cobrado una gran importancia y su uso se ha hecho muy común. Esta fuerte demanda de aplicaciones Web ha hecho que hayan nacido una serie de *frameworks* que facilitan su desarrollo.

Este texto presenta una aplicación Web estándar implementada con Struts, uno de los *frameworks OpenSource* con más aceptación de los disponibles en el mercado para el desarrollo de aplicaciones Web. Esta aplicación se completa con una interfaz de comunicación mediante servicios Web que, a partir del protocolo de comunicación SOAP, permite el intercambio de datos entre aplicaciones. Este modo de comunicación también utiliza Internet como medio de transporte.

Índice.

Índice de figuras.

1. Introducción

Actualmente las aplicaciones Web han cobrado mucha importancia en todos los ámbitos empresariales y comerciales. Internet se ha convertido en una herramienta muy común tanto en empresas como en hogares, lo que ha hecho que realizar compras a través de este medio sea una práctica cada vez más habitual. En las empresas, las aplicaciones Web han pasado a sustituir casi en su totalidad a las aplicaciones cliente-servidor, que necesitan estar instaladas en cada ordenador personal del usuario que va a utilizarlas. Las aplicaciones Web funcionan de forma similar independientemente del sistema operativo del usuario ya que, a diferencia de lo anterior, utilizan el navegador Web como cliente. Una aplicación Web se escribe una sola vez y funciona igual en todas partes.

Como consecuencia del éxito de las aplicaciones Web han nacido una serie de *frameworks* o marcos de trabajo cuya misión es facilitar el desarrollo de dichas aplicaciones así como enriquecer su funcionalidad de una manera muy sencilla. Por otro lado, también es muy frecuente que dos aplicaciones, elaboradas con la misma o distintas tecnologías, interactúen entre sí. En este ámbito los servicios Web han cobrado mucha fuerza en los últimos años, ya que permiten la interoperabilidad entre aplicaciones a través de Internet.

El objetivo central de este proyecto es mostrar las capacidades de uno de los *frameworks* de presentación punteros en la creación de aplicaciones Web mediante el desarrollo de una tienda *on-line* y ver que la implementación de servicios Web se puede realizar de una forma muy sencilla y dota de gran potencia a las aplicaciones.

Para conseguir este objetivo se ha comenzado con una labor de investigación y comprensión de las tecnologías involucradas y a continuación se han construido dos aplicaciones a partir de ellas. La aplicación principal consiste en una tienda de libros *on-line* desarrollada íntegramente con el *framework* Struts. Esta aplicación se complementa con otra destinada a los proveedores de la librería que les permite modificar el *stock* del almacén a través de servicios Web.

El texto se divide en dos partes principales:

La primera presenta las tecnologías empleadas en el desarrollo del proyecto. Se hace especial hincapié en las principales, Struts y SOAP, aunque también se describen otras tecnologías auxiliares que les sirven de apoyo y las complementan.

La segunda parte se centra en el diseño e implementación de una librería electrónica mediante el *framework*, Struts y de una aplicación basada en servicios Web que permite que los proveedores interactúen con la base de datos sin necesidad de acceder a la aplicación Web. En esta parte se muestran la arquitectura del sistema y las decisiones tomadas a la hora de realizar el diseño.

Esta segunda parte describe la arquitectura del sistema, las decisiones tomadas en la fase de diseño y la implementación de las dos aplicaciones que conforman este proyecto. También se explica el funcionamiento de estas aplicaciones mediante una serie de ejemplos de funcionamiento y escenarios de uso propuestos.

El proyecto desarrolla las funcionalidades principales de la tienda de libros, explotando al máximo las capacidades del *framework* empleado. No entra dentro del alcance de este estudio cubrir todas y cada una de las funcionalidades necesarias en una aplicación comercial.

2. Struts

2.1. Introducción

Struts es un marco de trabajo o *framework* de código abierto, que fue creado para permitir el desarrollo de aplicaciones Web basándose en las tecnologías Java Servlet y JavaServer Pages.

Struts proporciona a los desarrolladores una infraestructura unificada sobre la que pueden basar las aplicaciones Web, de tal forma que puedan centrarse en la creación de la aplicación de negocio en lugar de la infraestructura.

El marco de trabajo Struts fue creado por Craig R. McClanahan y donado a la Apache Software Foundation en el año 2000. Este proyecto cuenta en la actualidad con varios desarrolladores dedicados que son los responsables máximos de las decisiones dentro del proyecto.

Struts es uno de los proyectos más conocido y de más éxito de Apache Jakarta, una plataforma de creación de *software* de la Apache Software Foundation. Apache Jakarta cuenta también con otros proyectos de gran importancia, como son Ant, log4j y Tomcat, de los que se tratará más adelante en este documento.

2.2. Servlets Java

Un *servlet* Java es un objeto que se ejecuta dentro de un contenedor de s*ervlets* o un servidor de aplicaciones. Están especialmente diseñados para generar páginas Web de forma dinámica a partir de los parámetros de la petición enviada por el navegador.

Los *servlets* Java se han convertido en un pilar importante que permite ampliar y mejorar las aplicaciones Web utilizando la plataforma Java. Proporcionan un método independiente de la máquina y sistema sobre el que se desarrollan, basado en componentes, para crear aplicaciones Web.

Los *servlets* no se ven afectados por las limitaciones de rendimiento que habitualmente afectan a las aplicaciones CGI estándar. Los *servlets* son más eficaces porque crean un solo

proceso pesado y permiten que cada petición de usuario utilice un proceso más ligero, que se mantiene por medio de la máquina virtual de Java (JVM), para completar la petición. Las múltiples peticiones de usuario se pueden procesar por medio de la misma instancia de un *servlet*, ya que se invoca el método de servicio en el *servlet* y éste responde. Esta naturaleza multiproceso de los *servlet* es una de las principales razones por las que son más escalables que las aplicaciones CGI estándar.

El hecho de que los *servlets* estén escritos en Java implica que pueden utilizar todos los interfaces de programación de aplicaciones (Application Programming Interfaces o API) de Java, incluido Java DataBase Connectivity (JDBC), que permite las conexiones a bases de datos, y Enterprise JavaBeans (EJB).

Los *servlets* forman parte de JEE (Java Enterprise Edition), que es una ampliación de JSE (Java Standard Edition). Son objetos Java que implementan la interfaz `javax.servlet.Servlet` o heredan alguna de las clases más convenientes para un protocolo específico, por ejemplo `java.servlet.HttpServlet`. Al implementar esta interfaz el *servlet* es capaz de interpretar los objetos de tipo `HttpServletRequest` y `HttpServletResponse`, los que contienen la información de la página que invocó al *servlet*.

Los *servlets* no pueden ejecutarse directamente por un servidor Web, sino que requieren de un contenedor de *servlet*. Este contenedor está acoplado a una instancia determinada de un servidor Web y juntos cooperan para servir peticiones. Entre el contenedor Web y el *servlet* existe un contrato que determina cómo han de interactuar. Esta especificación se encuentra en los JSR (Java Specification Request) del JCP (Java Community Process).

2.3. JavaServer Pages

Las JavaServer Pages (JSP) son una extensión natural de la tecnología Java Servlet. Las páginas jsp son documentos de texto que tienen la extensión .jsp y que contienen una combinación de etiquetas estáticas HTML, XML y *scriptlets*.

Las etiquetas y los *scriptlets* encapsulan la lógica que genera el contenido de las páginas. Los archivos .jsp se preprocesan y se convierten en archivos .java. Estos archivos .java son compilados y generan los archivos .class que pueden ser ejecutados por el contenedor.

Los *scriptlets* son bloques de código Java insertados dentro de la página jsp. Hay diferentes opiniones acerca del uso de *scriplets* o de las bibliotecas de etiquetas para ejecutar lógica dentro

de las páginas jsp. Muchos desarrolladores consideran que se deben utilizar las bibliotecas de etiquetas personalizadas en lugar de los *scriplets*. Las razones fundamentales para esta creencia son:

- Los *scriplets* mezclan la lógica con la presentación.
- Los *scriplets* rompen la separación de roles.
- Los *scriptlets* hacen que las páginas jsp sean difíciles de leer y mantener.

Las etiquetas personalizadas, por otro lado, centralizan código en una sola ubicación y ayudan a mantener la separación de responsabilidades. También soportan el concepto de reutilización, ya que la misma etiqueta se puede insertar en múltiples páginas, mientras que la implementación reside en una única ubicación.

La tecnología JSP se ha convertido en una solución extremadamente popular para crear aplicaciones Web utilizando la plataforma Java. JSP ofrece numerosas ventajas frente a sus competidores:

- JSP es una especificación, no es un producto. Por ello los desarrolladores pueden elegir el enfoque que mejor se adapte a sus necesidades.
- Las páginas jsp se compilan, no se interpretan, lo que suele implicar un mejor rendimiento.
- Las páginas jsp soportan tanto *scripts* como acceso al lenguaje Java completo y se pueden ampliar por medio de etiquetas personalizadas.
- Las páginas jsp comparten el concepto WORA (Write Once, Run Anywhere) de la tecnología Java. Por lo tanto son portables a otros sistemas operativos y servidores Web.
- Las páginas jsp cuentan con JSP Actions, que son etiquetas XML que invocan funcionalidades en el servidor. Algunas de ellas son etiquetas estándar y otras han sido modificadas por los propios desarrolladores.

Desde la versión 2.0 de JSP se han incluido nuevas funcionalidades con el fin de aumentar la productividad, reutilización de código y separar de manera clara la presentación y la lógica de negocio, que normalmente es implementada por las clases Java. Se desarrolló Expression

Language (EL)[1] para hacer referencia a componentes Java de una manera limpia y elegante, evitando el uso de *scriptlets*. Además se simplificó la creación de bibliotecas de etiquetas personalizadas.

2.4. El patrón MVC

El patrón Modelo-Vista-Controlador (MVC) es un patrón de arquitectura *software* que separa los datos de una aplicación, la interfaz de usuario y la lógica de control en tres componentes distintos.

Modelo

Es el reponsable de la lógica de negocio. Contiene los datos y la funcionalidad de la aplicación. Es independiente de la representación de los datos.

Dependiendo del tipo de arquitectura que se utilice, la parte del modelo del patrón MVC puede adoptar muchas formas diferentes. En una aplicación de dos niveles, donde el nivel Web interactúa directamente con un almacén de datos, las clases del modelo pueden ser un conjunto de objetos Java. Estos objetos pueden ser obtenidos manualmente a partir de un `ResultSet` devuelto por una consulta de base de datos.

En una aplicación empresarial más compleja la parte del modelo del patrón MVC puede ser Enterprise JavaBeans, aunque el uso de esta especificación puede causar un importante impacto en el rendimiento.

Vista

La vista dentro del patrón MVC de nivel Web consta, por lo general, de páginas web HTML y JSP. Las páginas HTML se utilizan para servir contenido estático, mientras que las páginas jsp se pueden utilizar para servir contenido tanto estático como dinámico. La mayor parte del contenido dinámico se genera en el nivel Web. Sin embargo, algunas aplicaciones pueden requerir *javascript* por parte del cliente. Esto no interfiere con el concepto MVC.

HTML y JSP no son las únicas opciones para la vista. Se puede emplear, por ejemplo, WML en lugar de HTML. Puesto que la vista se desacopla del modelo, la aplicación puede soportar

[1] Expression Language es un lenguaje de *scripting* que permite acceder facilmente a los JavaBeans desde páginas jsp mediante una sintaxis sencilla.
[2] Un *scriptlet* es un trozo de código Java embebido dentro de una página jsp o HTML.

múltiples vistas, cada una de ellas para un tipo de cliente diferente, utilizando los mismos componentes del modelo.

Controlador

Es el responsable de controlar el flujo y estado de la entrada de datos por parte del usuario. La parte del controlador del diseño MVC de nivel Web es normalmente un *servlet* Java que realiza las siguientes funciones:

- Intercepta peticiones HTTP desde un cliente.
- Traduce cada petición en una operación específica de negocio a llevar a cabo.
- Invoca la operación de negocio o la delega a un manejador.
- Ayuda a seleccionar la siguiente vista que se ha de mostrar al cliente.
- Devuelve la vista al cliente.

El patrón Front Controller, que es parte de los patrones de diseño de J2EE (Java 2 Platform Entrerprise Edition), describe cómo se debería implementar un controlador de nivel Web. Puesto que todas las peticiones y respuestas del cliente pasan por el controlador, existe un punto centralizado de control para la aplicación Web. Esto es de gran ayuda cuando se añade una nueva funcionalidad. El código que normalmente se necesitaría situar en cada página jsp se puede situar en el *servlet* del controlador, que procesa todas las peticiones. El controlador también ayuda a desacoplar los componentes de presentación de las operaciones de negocio, haciendo más fácil el desarrollo.

2.5. El framework o marco de trabajo

Cómo se ha comentado anteriormente, Struts es un *framework* o marco de trabajo sobre el que los desarrolladores implementan sus aplicaciones. A continuación se mostrará con más detalle qué se entiende por marco de trabajo.

2.5.1. El concepto de marco de trabajo

Un marco de trabajo es un conjunto de clases e interfaces que cooperan para solucionar un tipo específico de problema de *software*. Tiene las siguientes características:

- Consta de múltiples clases o componentes, cada una de las cuales puede proporcionar una abstracción de algún concepto determinado.
- Define cómo estas abstracciones trabajan conjuntamente para solucionar un problema.
- Sus componentes son reutilizables.
- Un marco de trabajo organiza patrones en un nivel superior.
- Un buen marco de trabajo debe proporcionar un comportamiento genérico que puedan utilizar muchos tipos de aplicaciones diferentes.

Se debe diferenciar entre lo que es una biblioteca de *software* y un marco de trabajo. Una biblioteca de *software* consta de funciones o rutinas que una aplicación puede invocar. Un marco de trabajo, por el contrario, proporciona componentes genéricos y cooperativos que la aplicación amplía para proporcionar un conjunto determinado de funciones.

Los marcos de trabajo son diseñados con el intento de facilitar el desarrollo del *software*, permitiendo a los diseñadores y programadores pasar más tiempo identificando requerimientos de *software* que tratando con los tediosos detalles de bajo nivel de proveer un sistema funcional.

No obstante, hay quejas acerca de que el uso de marcos de trabajo añade código innecesario a la aplicación y que la proliferación de marcos de trabajo competitivos y complementarios provoca que el tiempo que antes se empleaba en programar y diseñar la aplicación ahora se utiliza en aprender a usarlos.

2.5.2. Alternativas a Struts

Aunque este proyecto está dedicado en gran medida al marco de trabajo Apache Struts, éste no es el único disponible. Existen algunos más, cada uno de ellos con seguidores y detractores.

Apache Struts cuenta con ventajas e inconvenientes sobre otros marcos de trabajo que cada desarrollador debe evaluar para elegir el marco de trabajo que se adapte mejor a sus necesidades.

Algunos de estos marcos de trabajo disponibles son los siguientes:

Cocoon

Apache Cocoon es un marco de trabajo basado en las tecnologías XML y XSLT. Ofrece un entorno flexible basado en la separación de competencias entre contenido, lógica y estilo.

Cocoon interactúa con la mayoría de las fuentes de datos, incluyendo sistemas de archivos RDBMS, LDAP, bases de datos XML nativas y fuentes de datos basadas en red. Permite adaptar el contenido a las capacidades de diferentes dispositivos, creando dinámicamente diferentes versiones de un mismo documento en diferentes formatos, como HTML, WML, PDF, SVG o RTF.

Cocoon es una buena alternativa si se busca un *framework* que tenga capacidades XML y XSLT, ya que, a diferencia de otros *frameworks* que proveen extensiones para habilitar el uso de estos lenguajes, XML y XSL forman parte del núcleo de Cocoon.

Turbine

Turbine es un *framework* que forma parte del proyecto Apache. Está basado en tecnología *servlet* y permite a los desarrolladores construir aplicaciones Web de una forma relativamente sencilla. Al igual que Struts, también sigue el patrón MVC, de forma que cuando recibe alguna petición comprueba si hay alguna acción asociada y, si es así, la ejecuta. Tras esto genera la página de respuesta, pero para ello usa varios módulos: el módulo de Layout, que configura la apariencia de la página, el módulo Screen, que genera el cuerpo y el módulo Navigation, que genera los controles de navegación entre páginas.

Incluye gran cantidad de servicios como control de acceso, persistencia, *logging*, internacionalización, etc.

Spring

Spring es un marco de trabajo *OpenSource* cuyo código fuente está bajo la licencia de Apache. Spring permite manejar patrones de diseño de una manera sencilla. Es muy modular, por lo que se pueden usar algunos de sus módulos de forma independiente. Una de sus principales características es que se integra fácilmente con otros *frameworks*, como JavaServer Faces o el propio Struts.

Spring proporciona un potente mecanismo de gestión de la configuración basada en JavaBeans, aplicando los principios de Inversión de Control (IoC), lo que hace que la configuración de aplicaciones sea rápida y sencilla. El contenedor de inversión de control de Spring instancia los Beans del sistema y les asigna sus dependencias. Para ello, necesita que mediante información de configuración se le indique dónde se encuentran dichos Beans.

Al igual que Struts, Spring también está basado en el patrón Modelo-Vista-Controlador.

JavaServer Faces

JavaServer Faces (JSF), como Struts y Spring, también cumple con el patrón MVC. Proporciona al desarrollador un conjunto de APIs para representar los componentes de una interfaz de usuario y administrar su estado, manejar eventos, validar las entradas, definir un esquema de navegación y dar soporte para internacionalización y accesibilidad.

Al ser posterior a Struts, JSF se ha nutrido de la experiencia de éste, mejorándolo en algunos aspectos.

JBoss Seam

JBoss Seam es un *framework* desarrollado por JBoss, una división de RedHat. Este proyecto combina los *frameworks* Enterprise JavaBeans 3 y JavaServer Faces. Introduce el concepto de contextos. Cada componente del *framework* está disponible dentro de un contexto. Dispone de un contexto conversacional que almacena todas las acciones del usuario hasta que éste sale del sistema o cierra el navegador. Se integra fácilmente con las bibliotecas de componentes JBoss RichFaces y ICEFaces. Para el desarrollo se facilitan las JBoss Tools, que son un conjunto de *plug-ins* diseñados para funcionar con el entorno de desarrollo Eclipse.

2.6. El nivel Web

El marco de trabajo Struts está basado en las tecnologías Java Servlet y en JavaServer Pages, por lo tanto depende de un contenedor Web. Por ello, a continuación se detalla cómo es esta arquitectura Web y dónde encaja Struts dentro de ella.

2.6.1. Arquitectura de una aplicación web

Las aplicaciones J2EE se pueden describir, en su mayoría, en términos de sus niveles. Esta separación de niveles puede ser una separación física, donde cada uno de ellos está situado en un recurso *hardware* distinto, o puede ser una separación puramente lógica. En este último caso, uno o varios niveles se colocan en el mismo recurso *hardware* y la separación existe en términos de componentes *software*.

La **Figura 2-1** ilustra los niveles que puede utilizar una aplicación Struts típica.

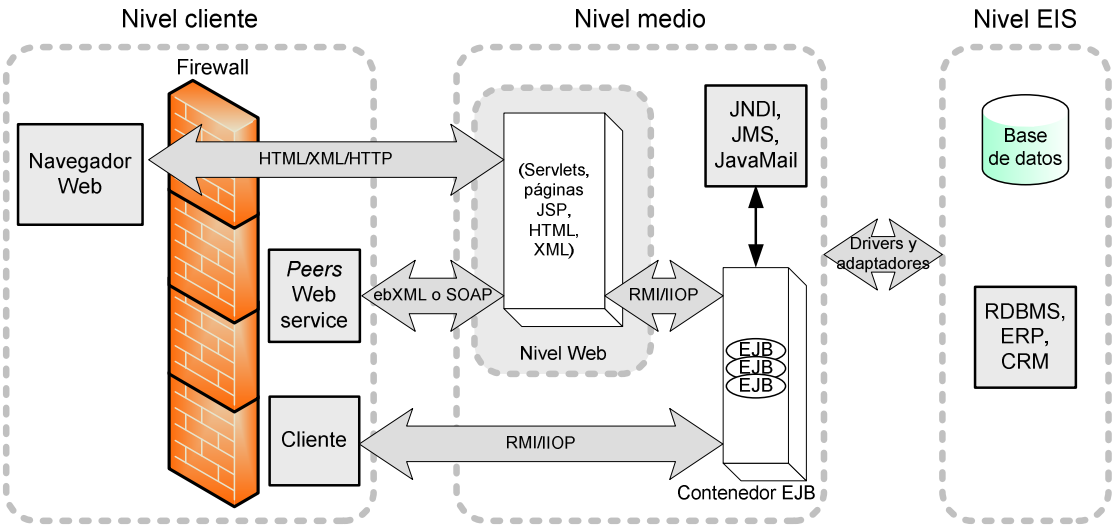


Figura 2-1. Niveles funcionales de aplicación

No todas las aplicaciones Struts contendrán todos los niveles ilustrados en la **Figura 2-1**. En aplicaciones pequeñas, el nivel medio puede constar de un contenedor Web que interactúa directamente con una base de datos en el nivel de Sistema de Información Empresarial (Enterprise Information System o EIS).

A continuación de detalla cada uno de estos niveles.

Nivel cliente

El nivel cliente proporciona un medio para que los usuarios interactúen con la aplicación. Esta interacción puede realizarse por medio de un navegador Web o a través de un interfaz de servicios Web, como puede ser el caso de SOAP, que se verá en el siguiente capítulo.

Con independencia del tipo de cliente que se utilice, la interacción incluye enviar una petición y recibir algún tipo de respuesta del nivel medio.

En el caso de Struts, el tipo más común de cliente es un navegador Web, aunque también es posible tener clientes como dispositivos inalámbricos o *applets* Java.

Nivel Web

La **Figura 2-1** muestra el nivel medio como el total del nivel Web más algún tipo de componente del servidor de aplicación, un contenedor EJB en este caso. A menudo, estos dos niveles se combinan y muchos servidores de aplicación incluyen la funcionalidad del nivel Web.

El nivel Web permite que el nivel cliente se comunique e interactúe con la lógica de la aplicación que reside en otros niveles. En aplicaciones Web pequeñas no es extraño que una parte o toda la lógica de aplicación resida en este nivel. En aplicaciones empresariales más grandes, el nivel Web actúa como un mero traductor y relaciona peticiones HTTP con invocaciones de servicio del nivel medio.

El nivel Web también es el responsable de gestionar el flujo de pantalla basándose en el estado de la aplicación y en el usuario.

Los componentes que residen en el nivel Web permiten que los desarrolladores amplíen la funcionalidad básica de un servicio Web. En el caso de Struts, esto se realiza mediante los componentes del marco de trabajo que se ejecutan en un contenedor *servlet*.

Nivel medio

El nivel medio a menudo se conoce como "nivel de aplicación" o "servidor". Esto es debido a que, a menudo existe un servidor de aplicación dentro de este nivel. No todas las aplicaciones Struts tienen este nivel, pero suele incluirse en grandes aplicaciones empresariales. Cuando está presente, el nivel Web se comunica con él utilizando alguna variación del protocolo de comunicación RMI (Remote Method Invocation).

Una de las principales finalidades de utilizar el nivel de aplicación es la de separar las responsabilidades de presentación de las del modelo y reglas de negocio para la aplicación. Hoy en día muchas aplicaciones Web utilizan servidores EJB para sus niveles de aplicación.

Nivel EIS

El nivel de Sistema de Información Empresarial o EIS proporciona el acceso a recursos tales como base de datos, mainframes, aplicaciones CRM o sistemas de planificación de recursos (Enterprise Resource Planning o ERP).

El nivel medio se comunica con los componentes en el nivel EIS utilizando protocolos específicos de recursos. Por ejemplo, para comunicarse con una base de datos relacional se utilizará un *driver* JDBC (Java DataBase Connection).

2.6.2. Contenedor

Existen varios tipos diferentes de contenedores, como los contenedores EJB, los contenedores Web, los contenedores *servlet*, etc. En general, un contenedor proporciona un

entorno de almacenamiento en el que ejecutar componentes de software. Los contenedores proporcionan servicios generales que pueden utilizar los componentes dentro del entorno, por lo que los desarrolladores no han de preocuparse por proporcionar estos servicios.

Un contenedor Web permite que se desarrollen y ejecuten dentro del contenedor *servlets*, JSP y otras clases Java.

Cuando se utilizan los servicios proporcionados por un contenedor los desarrolladores de componentes tienen que ceder cierto control del entorno al contenedor. Aunque cada proveedor puede implementar ciertas partes del contenedor de manera propietaria, deben seguir unas especificaciones para asegurarse que las aplicaciones se pueden portar.

2.6.3. Struts dentro de la arquitectura Web

Como se puede ver en la **Figura 2-2**, el marco de trabajo Struts reside en el nivel Web. Las aplicaciones Struts están albergadas en un contenedor Web y pueden hacer uso de los servicios proporcionados por el contenedor, como gestionar peticiones de clientes mediante los protocolos HTTP o HTTPS.

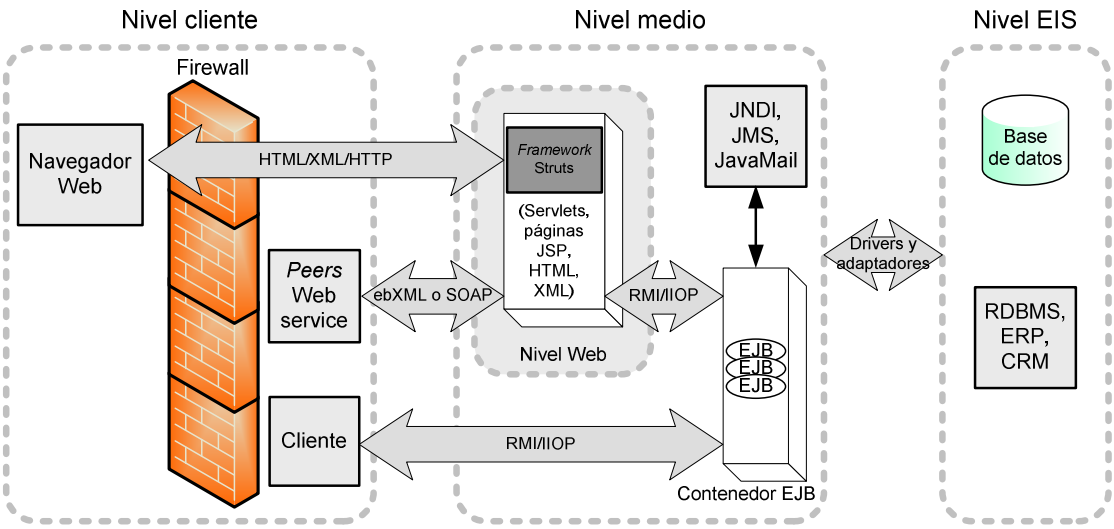


Figura 2-2. Struts en el nivel Web.

2.7. Configuración de aplicaciones en Struts

Una vez conocida la arquitectura de Struts, su utilidad y lo que proporciona al desarrollador, se va a analizar en profundidad el interior del marco de trabajo. Para ello, se considerará en primer lugar los ficheros de configuración que debe tener una aplicación Struts.

2.7.1. Ficheros de configuración

Struts utiliza dos tipos de archivos de configuración distintos, pero relacionados en cierta forma, que se deben configurar adecuadamente para que una aplicación funcione sin problemas.

Los archivos de configuración de Struts están basados en XML. Son dos archivos principales, el descriptor de despliegue de la aplicación Web, denominado *web.xml* y el archivo de configuración de Struts propiamente dicho, denominado *struts-config.xml*.

El fichero web.xml

El archivo de configuración *web.xml* es el descriptor de despliegue de la aplicación Web. Este archivo es necesario para todas las aplicaciones Web, no solamente para las aplicaciones creadas con el entorno de trabajo Struts. Sin embargo, en este fichero existe información de despliegue específica para Struts que se debe configurar cuando se crean aplicaciones en esta plataforma.

El paso más importante que se ha de realizar en el fichero *web.xml* es configurar el objeto `ActionServlet`, que recibirá todas las peticiones entrantes para la aplicación. Esto se lleva a cabo en dos pasos, primero se declara dentro del fichero la etiqueta *servlet* para configurar la instancia del *servlet* y posteriormente se mapea en el elemento `servlet-mapping`.

En el siguiente fragmento de código del fichero *web.xml* se ve cómo se configura la clase *servlet* utilizando la etiqueta correspondiente.

```
<web-app>
  <servlet>
      <servlet-name>action</servlet-name>
      <servlet-class>
      org.apache.struts.action.ActionServlet
      </server-class>
  </servlet>
  <servlet-mapping>
      <servlet-name>action</servlet-name>
      <url-pattern>*.do</url-pattern>
  </servlet-mapping>
</web-app>
```

El fichero struts-config.xml

El archivo *struts-config.xml* es un fichero descriptor que configura el marco de trabajo para formularios, acciones, mensajes, *plug-ins* y, lo que es más importante, el controlador de Struts.

A continuación se detallan cada uno de los elementos que se pueden configurar a través de este fichero.

data-sources: Permite establecer una fuente de datos para ser utilizada desde el marco de trabajo. Un origen de datos actúa como un *pool* de conexiones a una base de datos y proporciona un único punto de control.

```
<data-sources>
      <data-source type="org.apache.commons.dbcp.BasicDataSource">
         <set-property property="driverClassName"
              value="com.mysql.jdbc.Driver" />
         <set-property property="url"
              value="jdbc:mysql://localhost/elephant" />
         <set-property property="username" value="root" />
         <set-property property="password" value="secret" />
         <set-property property="maxActive" value="10" />
         <set-property property="maxWait" value="5000" />
         <set-property property="defaultAutoCommit" value="false" />
         <set-property property="defaultReadOnly" value="false" />
      </data-source>
</data-sources>
```

form-beans: Contiene las definiciones de los *beans* de la aplicación. Cada elemento `ActionForm` de la aplicación debe estar definido aquí. Los *beans* pueden ser dinámicos, esto es, sus propiedades se declaran en el archivo *struts-config.xml* o ser implementados mediante una clase `JavaBean`.

A continuación se muestra un ejemplo de *bean* no dinámico:

```
<fom-bean name="loginForm" type="pfc.ejemplo.login.formUser" />
```

global-exceptions: Permite configurar manejadores de excepciones de forma declarativa. Este ejemplo hace que si se lanza una excepción de fin de sesión, se reenvíe al usuario a la página de registro.

```
<global-exceptions>
  <exception key="expired.session"
              type="es.pfc.ExpiredSessionException" path="/login "/>
</global-exceptions>
```

global-forwards: Cuando una acción se ejecuta, se reenvía, en última instancia, a una vista. Esta vista suele ser una página jsp o HTML, pero podría ser cualquier otro tipo de recurso. En

lugar de hacer referencia a la vista directamente, Struts utiliza el concepto de reenvío para asociar un nombre lógico al recurso. De esta forma, en lugar de hacer referencia a *login.jsp,* una aplicación Struts puede hacer referencia a este recurso como el reenvío `login`, por ejemplo.

```
<global-forwards>
  <forward name="login" path="/login.do"/>
</global-forwards>
```

action-mappings: Describe un mapeado desde una ruta de petición específica a una clase `Action` correspondiente. El controlador selecciona un mapeado determinado al hacer coincidir la ruta de acceso del URI en la petición con el atributo `path` en uno de los elementos `action`.

A continuación se muestra un ejemplo:

```
<action path=”/login” type=”pfc.ejemplo.login.ActionUser” name=”loginForm”>
  <forward name=”sucess” action=”/pages/inicioAplicacion.jsp”/>
  <forward name=”error” action=”/pages/loginError.jsp”/>
</action>
```

controller: Se utiliza para definir parámetros específicos del controlador, como el tamaño del *buffer* de entrada, el tipo de codificación, etc.

En el bloque de código que se muestra a continuación se especifica el nombre completo de la clase Java que se utiliza para procesar peticiones.

```
<controller processorClass="org.apache.struts.tiles.TilesRequestProcessor"/>
```

message-resources: Especifica las características del paquete de recursos de mensaje que contiene los mensajes adaptados para una aplicación.

```
<message-resources parameter="java/MessageResources"/>
```

plug-in: Permite que las aplicaciones Struts creen recursos dinámicamente cuando se inician.

```
<plug-in className="org.apache.struts.validator.ValidatorPlugIn">
    <set-property property="pathnames"
        value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
```

2.8. El patrón MVC en Struts

Una vez se ha visto cómo funciona el patrón Modelo-Vista-Controlador en general para cualquier aplicación Web que se ajuste a este modelo, se muestra cómo es implementado por Struts.

2.8.1. El modelo Struts

El modelo de una aplicación representa sus datos de negocio. En el marco de trabajo Struts, la parte del modelo de una aplicación es bastante abierta. Es, de las tres partes del patrón MVC, en la que el desarrollador de una aplicación Struts tiene más libertad.

Permite al desarrollador crear su propia implementación del modelo de negocio o integrarse con otras tecnologías estándar de acceso a datos, como JDBC y EJB, o con otros *frameworks* como Hibernate.

2.8.2. La vista Struts

A continuación se detallan los componentes que conforman la vista dentro del marco de trabajo Struts. Aunque Struts también deja libertad al desarrollador para la implementación de la vista, provee de componentes que facilitan su desarrollo. Estos componentes proporcionan, entre otras cosas, soporte para internacionalización, aceptación de entrada de datos por parte del usuario, validación y gestión de errores.

En sentido general, una vista representa una muestra del modelo de dominio en una interfaz de usuario. Puede haber muchas vistas diferentes del mismo modelo. En el marco de trabajo Struts, las vistas se crean, normalmente, utilizando páginas jsp. Pero hay componentes adicionales que se pueden utilizar para mostrar las vistas junto con las páginas jsp, como son:

Documentos HTML

Aunque los documentos HTML generan solamente contenido estático, no están reñidos con el funcionamiento del marco de trabajo. No obstante, para aprovecharse de las características automáticas, especialmente las etiquetas personalizadas, hay que utilizar JSP en lugar de HTML.

Bibliotecas de etiquetas personalizadas JSP

Las etiquetas personalizadas desempeñan un rol importante dentro de una aplicación Struts. Aunque no es necesario que las aplicaciones las utilicen, algunos escenarios serían muy difíciles de programar sin ellas. Más adelante se verán con más detalle.

Javascript y hojas de estilo

El marco de trabajo no impide el uso de *javascript* dentro de una aplicación. Por el contrario, proporciona funcionalidad dentro de las bibliotecas de etiquetas para facilitar su uso.

En el caso de las hojas de estilo, Struts no limita en absoluto su uso. El desarrollador puede incluirlas en las páginas jsp y se mostrarán en el navegador como se haría con las páginas HTML.

Paquetes de recursos de mensaje

Los paquetes de recursos de mensaje son un componente muy importante para las aplicaciones Struts, ya que proporcionan un medio para soportar la internacionalización y ayudan a reducir el tiempo de mantenimiento y la redundancia de una aplicación.

Clases ActionForm

Por norma general, una aplicación Web debe aceptar la entrada de datos por parte del usuario. Cuando el usuario introduce estos datos y pulsa el botón adecuado en la aplicación, los valores se envían al servidor junto con la petición HTTP. Una vez en el servidor, estos datos se recogen y se pasan a otro componente que opera con ellos. En el caso en que los datos no fueran válidos, la aplicación debería regresar a la ubicación anterior, volver a mostrar uno o todos los valores introducidos por el usuario y mostraría el mensaje de error apropiado.

Desarrollar esta funcionalidad manualmente puede resultar una tarea tediosa. Por ello, Struts proporciona una manera muy sencilla de gestionar esta funcionalidad. Se basa en la clase `org.apache.struts.action.ActionForm`, que es el componente clave para realizar estas tareas.

La clase `ActionForm` se utiliza para capturar los datos de entrada de un formulario HTML y transferirlos a la clase `Action`, que forma parte del controlador y de la que se hablará más adelante. Puesto que los usuarios, en ocasiones, introducen datos no válidos, las aplicaciones necesitan disponer de un medio para almacenar los datos temporalmente de modo que se puedan

volver a mostrarlos cuando ocurre un error. La clase `ActionForm` actúa como un *buffer* que permite albergar el estado de los datos mientras se están validando.

Además, la clase `ActionForm` también actúa como cortafuegos para la aplicación, en el sentido de que ayuda a mantener los datos de entrada no válidos o sospechosos fuera del nivel de negocio hasta que hayan sido comprobados y aceptados.

Cuando los datos introducidos superan la validación de entrada, el objeto `ActionForm` se pasa al método `execute()` de la clase `Action`. Desde allí los datos se pueden recuperar del `ActionForm` y pasarlos al nivel de negocio.

No es necesario utilizar un `ActionForm` diferente para cada formulario de una aplicación. Una misma clase `ActionForm` se podría reutilizar en diferentes partes de la aplicación.

El principal problema que puede ocurrir con el uso de `ActionForm` es el amplio número de clases que se puede añadir a un proyecto, incluso aunque se compartan definiciones `ActionForm` en muchas páginas. Las clases adicionales dificultan la gestión y el mantenimiento. Por estas razones aparece una clase que es dinámica y evita la creación de objetos concretos `ActionForm` para su aplicación.

Este tipo de `ActionForm` dinámica se implementa por medio de la clase básica `org.apache.struts.action.DynaActionForm`, que amplía la clase `ActionForm`. Existen solamente tres diferencias entre la clase dinámica y la básica:

- **Las propiedades que define:** en el caso de la clase `DynaActionForm` las propiedades se definen en el fichero *struts-config.xml*. No es necesario implementar la clase Java correspondiente.
- **El método `validate()`:** no existe una manera sencilla de validar los datos utilizando `DynaActionForm`, para hacerlo se utiliza el `Validator`, que se verá más adelante.
- **El método `reset()`:** este método se invoca exactamente durante el procesamiento de la petición, al igual que para un objeto `ActionForm`. La única diferencia entre ambos radica en que, en este caso, se tiene menos control durante la invocación del método.

2.8.3. El controlador Struts

Los componentes del controlador son los responsables de detectar la entrada de datos del usuario, actualizar el modelo y seleccionar la siguiente vista que se ha de mostrar al cliente.

El controlador actúa como mediador entre la entrada de datos del cliente y el modelo, proporcionando una funcionalidad común, así como seguridad, conexión y otros servicios importantes ante cada petición del cliente. Además, como todas las peticiones se filtran a través del controlador, la vista se encuentra muy desacoplada de la lógica de negocio. La vista que se devuelve al cliente depende por completo del controlador, lo que hace que las aplicaciones sean mucho más flexibles.

Las tareas de las que se encarga el controlador son las siguientes:

- Interceptar las peticiones del cliente.
- Relacionar cada petición con una operación específica de negocio.
- Recopilar los resultados de la operación de negocio y ponerlos a disposición del cliente.
- Determinar la vista a mostrar al cliente basada en el estado actual y en el resultado de la operación de negocio.

El controlador utiliza varios componentes que son los responsables de estas tareas. Son los siguientes:

La clase ActionServlet

La clase `org.apache.struts.action.ActionServlet` actúa como un interceptor para la aplicación Struts. Todas las peticiones del cliente deben pasar por el *ActionServlet* antes de continuar hacia algún otro sitio en la aplicación.

Cuando una instancia de `ActionServlet` recibe una `HttpRequest`, bien sea a través del método `doGet()` o `doPost()`, se invoca al método `process()` de `ActionServlet` para gestionar dicha petición.

La clase RequestProcessor

El siguiente paso es llamar al método `process()` de la clase `org.apache.struts.action.RequestProcessor`. Lo llama la instancia `ActionServlet` y se le pasa la petición actual, además de los objetos de respuesta.

La clase Action

La clase `org.apache.struts.action.Action` es el corazón del marco de trabajo. Es el puente entre una petición de cliente y una operación de negocio. La clase `Action` dispone del método `execute()`, el que llevará a cabo las operaciones necesarias en la lógica de negocio con los datos obtenidos a través de la vista y devolverá el resultado correspondiente.

Una extensión de la clase `Action` es la clase `DispatchAction`, que permite que múltiples operaciones se agrupen en una sola clase, en vez de encontrarse distribuidas en múltiples clases `Action`.

La clase ActionForward

El método `execute()` de la clase `Action` devuelve como resultado un objeto `ActionForward`. La clase `ActionForward` representa una abstracción lógica de un recurso Web. Este recurso es, normalmente, una página jsp o un *servlet* Java.

2.9. Bibliotecas de etiquetas personalizadas para JSP

Las etiquetas personalizadas para JSP permiten a los desarrolladores ampliar las etiquetas disponibles más allá de las que JSP proporciona. Estas bibliotecas de etiquetas se pueden reutilizar en distintas aplicaciones.

El marco de trabajo Struts se aprovecha de la biblioteca de etiquetas de JSP, que contiene varias categorías de etiquetas personalizadas que ayudan a hacer la capa de presentación más manejable y reutilizable. Utilizando las bibliotecas de etiquetas personalizadas de Struts, los desarrolladores pueden interactuar con el resto del marco sin incluir código Java en las páginas jsp.

2.9.1. Etiquetas JSP personalizadas

Cuando se analiza un archivo HTML, el navegador determina cómo procesar y gestionar las etiquetas contenidas dentro del archivo basándose en un conjunto de estándares.

El objeto de las etiquetas JSP personalizadas es ofrecer al desarrollador la posibilidad de ampliar el conjunto de etiquetas que se pueden utilizar dentro de una página jsp. Con etiquetas HTML normales, el navegador contiene la lógica para procesar la etiqueta y mostrar el

resultado. Con etiquetas JSP personalizadas, la funcionalidad existe en una clase Java especial denominada manejador de etiquetas.

El manejador de etiquetas es una clase Java que lleva a cabo el comportamiento específico de la etiqueta. Implementa una de las posibles interfaces de etiquetas personalizadas, dependiendo del tipo de etiqueta que tenga que desarrollar. La clase del manejador tiene acceso a todos los recursos JSP tales como el objeto *PageContext* y los objetos de petición, respuesta y sesión. La etiqueta también se completa con la información de atributo, por lo que puede personalizar su comportamiento basándose en los valores del atributo.

2.9.2. Ventajas de usar etiquetas personalizadas

El uso de etiquetas personalizadas en lugar de *scriptlets*[2] y código Java en las páginas jsp proporciona muchas ventajas:

- Las etiquetas son reutilizables, lo que ahorra tiempo de desarrollo.
- Las etiquetas se pueden personalizar utilizando atributos, tanto de forma estática como dinámica.
- Las etiquetas tienen acceso a todos los objetos disponibles para la página jsp, incluidas las variables de petición, respuesta y sesión.
- Las etiquetas se pueden anidar, lo que permite interacciones más complejas dentro de una página jsp.
- Las etiquetas hacen más legible una página jsp.

En términos generales, emplear etiquetas personalizadas JSP ayuda a ampliar el concepto de reutilización, ya que el comportamiento se implementa en una única ubicación y es invocado a través de múltiples páginas jsp.

2.9.3. Bibliotecas de etiquetas incluidas con Struts

Como se ha ido viendo, el marco de trabajo Struts proporciona un conjunto bastante amplio de componentes. Además, también incluye un conjunto de etiquetas diseñadas para interactuar con el resto del marco. Las etiquetas personalizadas incluidas con Struts se agrupan en cinco bibliotecas distintas, que se muestran a continuación.

[2] Un *scriptlet* es un trozo de código Java embebido dentro de una página jsp o HTML.

HTML

La biblioteca de etiquetas HTML de Struts contiene etiquetas para crear formularios de entrada de datos HTML, al igual que otras etiquetas de utilidad en la creación de interfaces de usuario basadas en HTML.

Bean

Las etiquetas que forman parte de la biblioteca Bean se utilizan para acceder a JavaBeans y a sus propiedades asociadas, al igual que para definir nuevos *beans* que son accesibles para el resto de la página a través de variables de *script* y atributos de ámbito de página. Además, también se facilitan mecanismos para crear nuevos *beans* basados en los valores de las *cookies* de petición, encabezados y parámetros.

Logic

La biblioteca de etiquetas Logic contiene etiquetas útiles para gestionar la creación condicional de texto de salida, iterar sobre colecciones de objeto para una generación dinámica de texto de salida y para la gestión de flujo de la aplicación.

Nested

En ocasiones, cuando se intentan anidar etiquetas dentro de otras, la etiqueta de dentro puede tener ciertas dependencias con la de fuera, lo que puede impedir que se muestren los datos dinámicos. Las etiquetas Nested se crearon para solucionar este problema y enriquecer las bibliotecas de etiquetas Struts.

Las etiquetas de la biblioteca Nested se utilizan de la misma manera que las de las versiones no anidadas. Existen diferencias menores para que las etiquetas sepan cuando están anidadas dentro de otras y puedan, por tanto, acceder a las propiedades de la etiqueta padre. A excepción de estas pequeñas diferencias, son muy similares a las etiquetas no anidadas.

2.10. Tiles

La siguiente sección muestra un conjunto especial de etiquetas que se integra perfectamente con Struts. Este conjunto es el marco de trabajo Tiles, que permite definir plantillas o *templates* y *layouts* para las páginas jsp.

2.10.1. Descripción

Las Tiles son un conjunto de etiquetas que permiten organizar y ensamblar el contenido de las páginas jsp. Permiten definir plantillas que se utilizarán dentro de las páginas jsp.

Con esto se evita la duplicación de código de lenguaje de marcado dentro de una aplicación Web. Por ejemplo, si en una aplicación todas las páginas jsp tienen la misma cabecera y el mismo pie de página, no es necesario incluir el código de esas partes en todas las páginas jsp, sino que se pueden crear plantillas que se insertan en las páginas necesarias.

Además de evitar la redundancia de código, también facilita el mantenimiento de la aplicación, puesto que, en este ejemplo, un cambio en la cabecera no implica un cambio en cada una de las páginas, sino un único cambio en la plantilla.

2.10.2. Instalar y configurar Tiles

El *framework* Tiles viene incluido con la distribución Struts. Las bibliotecas necesarias para su funcionamiento son:

- tiles.jar.
- commons- digester.jar.
- commons-beanutils.jar.
- commons-collection.jar.
- commons-logging.jar.

Además, como cualquier otra biblioteca de etiquetas JSP, hay que incluirla en el fichero *web.xml*:

```
<taglib>
  <taglib-uri>/WEB-INF/tiles.tld</taglib-uri>
  <taglib-location>/WEB-INF/tiles.tld</taglib-location>
</taglib>
```

Si se desean utilizar las definiciones de Tiles, es necesario también configurar como *plug-in* lo siguiente en el fichero *struts-config.xml*:

```
<plug-in className="org.apache.struts.tiles.TilesPlugin" >
  <set-property property="definitions-config"
      value="/WEB-INF/tiles-defs.xml" />
  <set-property property="definitions-debug" value="2" />
  <set-property property="definitions-parser-details" value="2" />
  <set-property property="definitions-parser-validate" value="true" />
```

```
</plug-in>
```

2.10.3. Definiciones

Las etiquetas Tiles se pueden utilizar como etiquetas normales o como definiciones. Las definiciones se declaran en el fichero *tiles-defs.xml*.

Una característica muy potente de las definiciones es la capacidad de herencia de las mismas. De este modo, es posible crear una definición base y dejar que las demás hereden de ésta. Los atributos del padre son heredados por la definición hija, pero se pueden sobrescribir algunos de ellos, según se necesite.

Las definiciones también pueden ser usadas en las clases *ActionForward*.

2.11. El marco de trabajo Validator

2.11.1. Descripción

Cada aplicación tiene la responsabilidad de asegurarse de que los datos de entrada sean datos válidos. Normalmente, en aplicaciones que usan una base de datos no se puede permitir cualquier dato de entrada. Los campos de entrada han de tener restricciones sobre los valores permitidos, ya que no validar estos datos antes de operar con ellos puede provocar graves problemas de seguridad. Además, operar con datos erróneos puede desvirtuar el comportamiento de la aplicación.

La validación de estos datos de entrada puede resultar una tarea tediosa, especialmente en aplicaciones muy grandes, con muchas páginas. Además, la validación de los datos suele ser muy redundante, ya que el mismo tipo de dato se puede comprobar en un amplio número de páginas jsp, como fechas, direcciones de correo, etc.

Para facilitar esta tarea, Struts se integra perfectamente con el marco de trabajo Validator que permite establecer reglas de validación de forma declarativa. Este marco de trabajo forma parte del proyecto Jakarta Commons.

Los beneficios de usar el marco de trabajo Validator son los siguientes:

- Hay una única definición de las reglas de validación para una aplicación.

- Las validaciones por parte del cliente y del servidor pueden ser definidas en una única localización.
- Modificar validaciones existentes o añadir nuevas se hace de forma sencilla.
- Soporta internacionalización.
- Soporta expresiones regulares.

2.11.2. Configuración del Validator

Para configurar las reglas de validación se utilizan principalmente dos ficheros de configuración.

Validation-rules.xml

Este fichero actúa como una plantilla y contiene las definiciones disponibles para una aplicación. Se debe almacenar en la carpeta *WEB-INF* del proyecto. El elemento más importante del fichero es el contenido por la etiqueta `<validator>`. Hay uno por cada regla usada por la aplicación.

A continuación se muestra un ejemplo de este fichero:

```
<form-validation>
 <global>
  <validator
     name="required"
     classname="org.apache.struts.util.StrutsValidator"
     method="validateRequired"
     methodparams="java.lang.Object,
                   org.apache.commons.validator.ValidatorAction,
                   org.apache.commons.validator.Field,
                   org.apache.struts.action.ActionErrors,
                   javax.servlet.http.HttpServletRequest"
     msg="errors.required"/>

  <validator name="minlength"
     classname="org.apache.struts.util.StrutsValidator"
     method="validateMinLength"
     methodparams="java.lang.Object,
                   org.apache.commons.validator.ValidatorAction,
                   org.apache.commons.validator.Field,
                   org.apache.struts.action.ActionErrors,
                   javax.servlet.http.HttpServletRequest"
     depends="required"
     msg="errors.minlength"/>
 </global>
</form-validation>
```

Validation.xml

El segundo fichero de configuración describe qué reglas de validación descritas en el archivo *validation-rules.xml* han de ser usadas por un *ActionForm* en concreto.

El siguiente código ofrece un ejemplo de este fichero:

```
<form-validation>
 <formset>
  <form name="checkoutForm">
    <field
      property="firstName"
      depends="required">
      <arg0 key="label.firstName"/>
    </field>

    <field
      property="lastName"
      depends="required">
      <arg0 key="label.lastName"/>
    </field>
  </form>
 </formset>
</form-validation>
```

Además, Validator dispone de un fichero de recursos en el que se especifica la información textual que se va a mostrar al usuario. Reduce considerablemente el texto redundante en la aplicación.

A continuación se muestra un ejemplo con los textos que hay por omisión, pero el desarrollador puede definir cuantos crea necesario o modificar los existentes.

```
errors.required={0} is required.
errors.minlength={0} can not be less than {1} characters.
errors.maxlength={0} can not be greater than {1} characters.
errors.invalid={0} is invalid.
```

Para que Validator funcione conjuntamente con el marco de trabajo Struts, se debe vincular con el mismo a través del fichero *web.xml*. Hay que añadir la siguiente entrada a este fichero:

```
<plug-in classname="org.apache.struts.validator.ValidatorPlugIn">
  <set-property
    property="pathnames"
    value="/WEB-INF/validator-rules.xml,/WEB-INF/validation.xml"/>
</plug-in>
```

2.12. Internacionalización

La internacionalización es el proceso de diseño de software empleado para permitir el uso de múltiples idiomas en la aplicación, sin necesidad de rediseñarla.

Struts soporta internacionalización a través de los ficheros de recursos y las bibliotecas de etiquetas personalizadas. Se pueden definir los textos a visualizar en el fichero de recursos que luego podrán ser usados en las JSP.

Las cadenas de texto del idioma empleado por defecto se guardan en el fichero *ApplicationRessources.properties*. Además, pueden definirse ficheros de recursos adicionales para el resto de idiomas.

3. SOAP

3.1. Introducción

En este capítulo se explicará qué es un servicio Web y lo que aporta al desarrollo de aplicaciones Web. Después de ver los tipos de servicios Web disponibles, se mostrará uno de ellos, el protocolo SOAP, que ha sido el utilizado en la elaboración de este proyecto.

3.2. Servicios Web

3.2.1. Descripción

Un servicio Web (en inglés Web Service) es un conjunto de protocolos y estándares que permiten, a través de Internet, el intercambio de datos entre aplicaciones. Para ello, utiliza un sistema de mensajería basado en XML estándar, independiente del sistema operativo o el lenguaje de programación.

Esta interoperabilidad se consigue mediante la adopción de estándares abiertos. Las organizaciones OASIS y W3C son las responsables de la arquitectura y reglamentación de los servicios Web. Para mejorar la interoperabilidad entre distintas implementaciones de servicios Web, se ha creado el organismo WS-I, encargado de desarrollar de manera más exhaustiva estos estándares.

Los servicios Web se basan en HTTP sobre TCP en el puerto 80. Generalmente las organizaciones protegen sus redes mediante *firewalls* que filtran y bloquean gran parte del tráfico de Internet, excepto el puerto 80, que corresponde al Web. Como los servicios Web se envían por este puerto, no resultan bloqueados.

Los servicios Web resultan muy prácticos porque aportan independencia entre la aplicación que usa el servicio Web y el propio servicio. De esta forma, cambios a lo largo del tiempo efectuados en uno, no deben afectar al otro. Esta flexibilidad es muy importante, puesto que cada vez es más común que las grandes aplicaciones se construyan a partir de componentes distribuidos más pequeños.

3.2.2. Arquitectura de un servicio Web

Hay dos formas distintas de ver la arquitectura de los servicios Web. La primera visión tiene en cuenta los distintos roles que entran en juego en un servicio, mientras que la segunda considera fundamentalmente la pila de protocolos que lo integran.

En primer lugar se describirá la arquitectura de un servicio Web a partir de sus diferentes roles. Hay tres roles principales, que son:

- **Proveedor de servicio**: Es el proveedor del servicio Web. Implementa el servicio y lo hace accesible a través de Internet.
- **Cliente**: El cliente es cualquier elemento que utilice el servicio Web. Lo hace abriendo una conexión de red al servicio y enviándole la petición XML.
- **Registro del servicio**: Este registro proporciona un lugar donde los desarrolladores pueden publicar nuevos servicios o localizar servicios existentes.

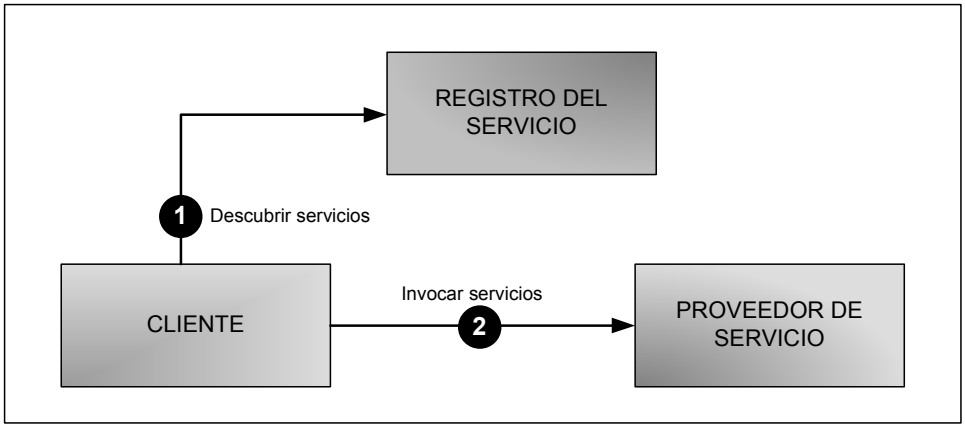


Figura 3-1. Arquitectura de un servicio Web.

La otra manera de ver la arquitectura Web se centra en los protocolos que la conforman. Esta pila de protocolos tiene cuatro capas principales:

- **Capa de transporte**: se encarga del transporte de mensajes entre aplicaciones. Actualmente esta capa incluye los siguientes protocolos: HyperText Transfer Protocol (HTTP), Simple Mail Transfer Protocol (SNMP), File Transfer Protocol (FTP) y otros protocolos nuevos, como Blocks Extensible Exchange Protocol (BEEP).

- **Mensajería XML**: esta capa es la responsable de codificar los mensajes en el formato XML apropiado que permite que sean interpretados en el destino. Incluye XML-RPC y SOAP.
- **Descripción del servicio**: es la capa que describe el interfaz público de un servicio Web. Es una descripción basada en XML de los requisitos funcionales necesarios para establecer una comunicación con los servicios Web. Actualmente esto se lleva a cabo mediante WSDL (Web Service Description Language).
- **Publicación del servicio**: centraliza los servicios en un registro común y facilita así su publicación y búsqueda. Esto es controlado por el UDDI (Universal Description Discovery and Integration).

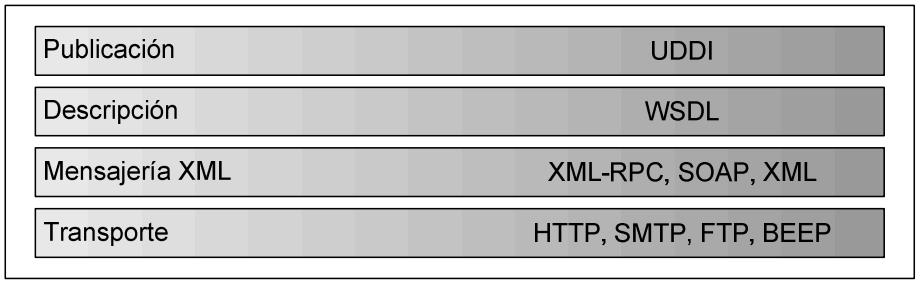


Figura 3-2. Capas de un servicio Web.

3.2.3. Ventajas en el uso de los Servicios Web

El uso de servicios Web aporta múltiples ventajas en el desarrollo de aplicaciones Web, como las siguientes:

- Aportan interoperabilidad entre aplicaciones de *software* independientemente de sus propiedades o de la arquitectura sobre la que se instalen.
- Los servicios Web fomentan los estándares y protocolos basados en texto, que hacen más fácil acceder a su contenido y entender su funcionamiento.
- Al apoyarse en HTTP, los servicios Web pueden aprovecharse de los sistemas de seguridad *firewall* sin necesidad de cambiar las reglas de filtrado.
- Permiten que servicios y *software* de diferentes compañías ubicadas en diferentes lugares geográficos puedan ser combinados fácilmente para proveer servicios integrados.

- Permiten la interoperabilidad entre plataformas de distintos fabricantes por medio de protocolos estándar.

3.2.4. Inconvenientes en el uso de los servicios Web

Algunos de los mayores inconvenientes de los servicios Web se muestran a continuación:

- Para realizar transacciones no pueden compararse en su grado desarrollo con los estándares abiertos de computación distribuida como CORBA (Common Object Request Broker Architecture).
- Su rendimiento es inferior al de otros modelos de computación distribuida, tales como RMI (Remote Method Invocation), CORBA o DCOM (Distributed Component Object Model). Este es uno de los inconvenientes derivados de adoptar un formato basado en texto.
- Al apoyarse en HTTP pueden esquivar las medidas de seguridad basadas en *firewall* cuyas reglas de filtrado tratan de bloquear o auditar la comunicación entre programas a ambos lados de la barrera.
- Existe poca información acerca de servicios Web para algunos lenguajes de programación.

3.3. SOAP

3.3.1. Introducción

SOAP corresponde a las siglas Simple Object Access Protocol. Se trata de un protocolo de comunicación basado en paso de mensajes. Está diseñado para el intercambio de información en arquitecturas distribuidas.

A diferencia de otros protocolos como RMI, CORBA o COM, que son binarios, SOAP usa el código fuente en XML. Esto implica que sea muy fácil su comprensión por parte de los humanos pero también hace que los mensajes resultantes sean más largos. No obstante se trata de un protocolo relativamente ligero.

SOAP es un marco extensible y descentralizado que permite trabajar sobre múltiples pilas de protocolos de redes informáticas. Los procedimientos de llamadas remotas pueden ser modelados en forma de varios mensajes SOAP interactuando entre sí.

3.3.2. El mensaje SOAP

Un mensaje SOAP está compuesto por tres partes diferenciadas:

Envelope (envoltura): es el elemento raíz del mensaje. Describe el contenido del mensaje y contiene información para procesarlo.

Header (encabezado): es la información de identificación del contenido. Consiste en un grupo de reglas de codificación de los tipos de datos que se pueden instanciar. Esta es una de las características más importantes de SOAP, su extensibilidad.

Body (cuerpo): es el contenido del mensaje. Una convención para representar las llamadas y las respuestas a procedimientos remotos.

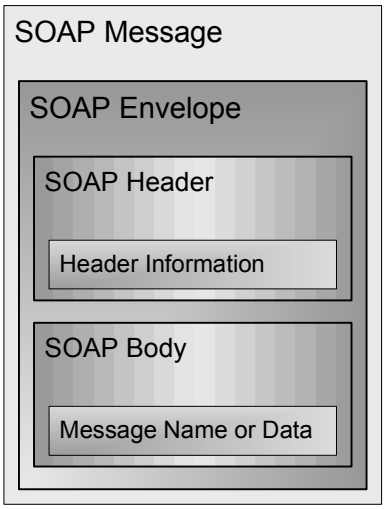


Figura 3-3. Partes del mensaje SOAP.

A continuación vemos un ejemplo de mensaje SOAP en XML:

```
<env:Envelope xmlns:env="http://www.w3.org/2003/05/soap-envelope">
 <env:Header>
  <n:alertcontrol xmlns:n="http://example.org/alertcontrol">
   <n:priority>1</n:priority>
   <n:expires>2001-06-22T14:00:00-05:00</n:expires>
  </n:alertcontrol>
 </env:Header>
 <env:Body>
```

```
  <m:alert xmlns:m="http://example.org/alert">
   <m:msg>Pick up Mary at school at 2pm</m:msg>
  </m:alert>
 </env:Body>
</env:Envelope>
```

En este ejemplo se puede apreciar, en formato XML, cómo se cumple con la estructura de la **Figura 3-3**. El elemento raíz del documento es el elemento `envelope`, que contiene dos subelementos, `body` y `header`. El elemento `header` o *cabecera* es opcional, pero si existe ha de ser el primer elemento hijo del `envelope`, con el elemento `body` o `cuerpo` a continuación.

El cuerpo contiene la carga de datos del mensaje y la cabecera los datos adicionales que no pertenecen necesariamente al cuerpo del mensaje.

3.3.3. Escenarios básicos de uso

Un mensaje SOAP es, fundamentalmente, una comunicación en un solo sentido entre nodos SOAP, de un remitente SOAP a un destinatario SOAP. Se espera que los mensajes SOAP sean combinados por las aplicaciones para implementar patrones de interacción más complejos, desde la petición-respuesta a múltiples intercambios conversacionales de ida y vuelta.

Escenario 1: Ampliación de stock en la librería por parte de la editorial

A continuación, se muestra un escenario básico en el que se describe una petición de ampliación de *stock* por parte de una editorial. En este escenario, una editorial solicita a una librería electrónica el aumento de *stock* de un determinado libro. La información intercambiada se realiza en forma de mensajes SOAP.

El destinatario último de un mensaje SOAP enviado desde la editorial es la librería, pero es posible que el mensaje SOAP pueda ser encaminado a través de uno o varios nodos SOAP intermediarios que actúen de algún modo sobre el mensaje.

El mensaje SOAP correspondiente a la solicitud de aumento de *stock* de un libro y su correspondiente respuesta se muestran a continuación:

Mensaje enviado al servidor

```
<?xml version='1.0' encoding='UTF-8'?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
      <ns1:updateStockRequest
          xmlns:ns1="http://www.example.org/editorialSchema/messages">
```

```
          <ISBN>8441319677</ISBN>
          <Cantidad>5</Cantidad>
      </ns1:updateStockRequest>
  </soapenv:Body>
</soapenv:Envelope>
```

Mensaje respuesta del servidor

```
<?xml version='1.0' encoding='UTF-8'?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
      <ns1:updateStockResponse
  xmlns:ns1="http://www.example.org/editorialSchema/messages">
          <status>
Se ha actualizado correctamente el Stock del libro 8441319677.
          </status>
      </ns1:updateStockResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

Como se puede observar, se ha utilizado el protocolo SOAP para invocar un método residente en la aplicación que contiene la librería. Los datos necesarios para realizar una llamada RPC son los siguientes:

- La dirección del nodo SOAP destino.
- El nombre del método.
- Los argumentos que han de ser pasados al método como parámetro.
- Una separación clara de los argumentos utilizados para identificar el recurso Web que es el destino real de la RPC, para diferenciarlos de aquellos que transportan datos o información de control.
- El patrón de intercambio de mensajes que será empleado para transportar la RPC junto con la identificación del método Web (GET o POST) a utilizar.
- Opcionalmente, los datos que deben ser transportados como partes de bloques de encabezado SOAP.

Escenario 2: Gestión de los errores

SOAP proporciona un modelo para la gestión de situaciones en las que surgen errores durante el proceso envío de un mensaje. SOAP distingue entre las condiciones que suponen un error y tiene la habilidad de señalar el error al remitente del mensaje equivocado o a otro nodo.

A continuación se detalla un ejemplo de mensaje de error en el que se indica que el libro, cuya cantidad se desea actualizar, no existe en la base de datos.

```
<?xml version='1.0' encoding='UTF-8'?>
<soapenv:Envelope
  xmlns:soapenv="http://schemas.xmlsoap.org/soap/envelope/">
  <soapenv:Body>
      <ns1:updateStockResponse
  xmlns:ns1="http://www.example.org/editorialSchema/messages">
          <status>
El proceso de inserción ha fallado para el Libro 84013357449
          </status>
      </ns1:updateStockResponse>
  </soapenv:Body>
</soapenv:Envelope>
```

3.3.4. Modelo de procesamiento SOAP

Una vez detallado cómo es un mensaje SOAP, se ofrecerá una visión general del modelo de procesamiento de SOAP.

Un mensaje SOAP puede pasar por uno o más intermediarios antes de ser procesado. Por ejemplo, puede haber un *proxy* SOAP entre una aplicación cliente y el servicio SOAP destino. El modelo de procesamiento describe las acciones tomadas por un nodo SOAP al recibir un mensaje.

Lo primero que ha de realizar un nodo SOAP es analizar que las partes del mensaje que acaba de recibir corresponden a un mensaje SOAP y que este mensaje esta bien formado.

El atributo *role*

El procesamiento posterior de los bloques de encabezado y el cuerpo depende del papel o rol (*role*, en inglés) asumido por el nodo SOAP para el procesamiento de un mensaje dado. El atributo `role` es opcional y puede estar presente en un bloque de encabezado. Indica el papel que juega el destinatario de ese bloque de encabezado. Se requiere que un nodo SOAP procese un bloque de encabezado si asume el papel identificado por el valor del URI.

Existen tres roles estandarizados, que son:

none: este rol implica que ningún nodo SOAP debería procesar su contenido. No obstante, un nodo podría necesitar examinarlo si el contenido está formado por datos que son referenciados por otro encabezado dirigido a un nodo SOAP particular.

next: cada nodo SOAP que reciba un bloque de encabezado que contenga el atributo `env:role` con valor `next` debe ser capaz de procesar los contenidos del elemento, ya que ese es el rol estandarizado que cada nodo SOAP debe poder asumir.

ultimateReceiver: el rol `ultimateReceiver` puede estar declarado explícitamente o no. Si no se define ningún rol se asume que es `ultimateReceiver`. El elemento `body` siempre va dirigido al nodo que asume este rol. En este sentido, el cuerpo es sólo como un bloque de encabezado dirigido al destinatario final pero es diferenciado para permitir que los nodos SOAP puedan pasarlo por alto si asumen otros papeles diferentes al del destinatario final.

El atributo *mustUndertand*

Después de que un nodo SOAP haya identificado correctamente los bloques de encabezado que han sido destinados a sí mismo utilizando el atributo `role`, el atributo adicional, `mustUnderstand` determina acciones de procesamiento posteriores que deben tomarse.

Para asegurar que los nodos SOAP no ignoran bloques de encabezado importantes para el propósito general de la aplicación, si el atributo `mustUnderstand` tiene el valor `true`, el nodo SOAP destinatario debe procesar dicho bloque de acuerdo con su especificación.

Un valor `true` para el atributo `mustUnderstand` significa que el nodo SOAP debe procesar el encabezado con la semántica descrita en la especificación del encabezado en sí, o generar un error SOAP. El procesamiento del encabezado de forma apropiada puede incluir la eliminación del encabezado de cualquier mensaje SOAP generado, reinsertando el encabezado con el mismo u otro valor, o insertando un nuevo encabezado. La incapacidad de procesar un encabezado obligatorio requiere que todo proceso posterior del mensaje SOAP se interrumpa, y que se genere un error SOAP. Si esto ocurre, el mensaje no se continúa enviando hacia su destino.

El atributo *relay*

SOAP define otro atributo opcional para bloques de encabezado llamado `relay`, de tipo *booleano*, que indica si un bloque de encabezado dirigido a un intermediario SOAP debe ser transmitido si no es procesado.

3.3.5. Ventajas en el uso de SOAP

Algunas de las ventajas en el uso de SOAP son las siguientes:

No está asociado con ningún lenguaje: SOAP no especifica ningún API, por lo que la implementación de la misma se deja al lenguaje de programación.

No se encuentra fuertemente asociado a ningún protocolo de transporte: la especificación de SOAP no describe cómo se deberían asociar los mensajes de SOAP con HTTP. Al no ser éstos más que un mensaje XML pueden ser transportados a través de cualquier protocolo capaz de transmitir texto.

No está atado a ninguna infraestructura de objeto distribuido: la mayoría de los sistemas de objetos distribuidos se pueden extender, y algunos de ellos ya se encuentran extendidos para que admitan SOAP.

Aprovecha los estándares existentes en la industria: los principales contribuyentes a la especificación de SOAP evitaron intencionadamente reinventar lo que ya era conocido. En lugar de ello, optaron por extender los estándares existentes para que coincidieran con sus necesidades.

Permite la interoperabilidad entre múltiples entornos: como se acaba de indicar, SOAP se desarrolló sobre los estándares existentes en la industria, por lo que las aplicaciones que se ejecuten en plataformas con dichos estándares se podrán comunicar con otras plataformas que también interpreten estos estándares.

3.3.6. Implementaciones de SOAP

Cada vez hay un número mayor de implementaciones del protocolo SOAP, algunas de ellas desarrolladas en Java. Varias son de uso libre y otras, por el contrario, pertenecen a una organización. Cada desarrollador ha de considerar qué implementación del protocolo se adapta a sus necesidades.

A pesar de que son muchas las implementaciones disponibles, en el desarrollo de este proyecto se ha utilizado la implementación de Apache de SOAP, de la que se hablará a continuación.

Apache SOAP: Axis y Axis2

Dos de los proyectos más importantes de la Apache Software Foundation para el uso de WebServices son los conocidos como Axis y Axis2. Estos proyectos consisten en una implementación en Java de la especificación del protocolo SOAP.

Para la elaboración de este proyecto se ha elegido el proyecto Axis2, que es un rediseño de Axis que soporta SOAP 1.1 y SOAP 1.2, entre otras versiones de servicios Web. Además es más eficiente, más modular y está más orientado a XML que Axis.

Algunas de las características de esta implementación que han ayudado en la elección a la hora de elaborar el proyecto son:

- Es muy estable.
- Soporta gran parte de los contenidos de la especificación.
- El código fuente está disponible para el desarrollador.
- Es libre, ya que está bajo la licencia GPL.
- Su instalación y el desarrollo de aplicaciones es sencilla.

Otras implementaciones

Como se ha indicado, cada desarrollador ha de utilizar la implementación del protocolo que le sea más conveniente. Por ello a continuación se mencionan algunas de las disponibles.

Una de las implementaciones de SOAP desarrolladas en Java es GLUE. Está desarrollada por una compañía llamada The Mind Electric y puede ser hospedada en cualquier servidor que soporte *servlets*, además de que puede ejecutarse como una aplicación única usando su propio servidor HTTP.

Otra de las implementaciones es la desarrollada por Microsoft para su entorno .NET. Permite desarrollar aplicaciones Java que usen SOAP para comunicarse con servicios .NET.

3.4. Apache Axis2

Como ya se ha dicho en el apartado anterior, Apache Axis 1.X es una implementación del protocolo SOAP. Axis2 nació en agosto de 2004 como un rediseño de Axis. Hay que destacar su mejor rendimiento, así como su sistema de módulos que permite, de forma sencilla, añadir nuevas funcionalidades y soportar futuras especificaciones sobre servicios Web.

Actualmente se puede considerar que esta implementación es muy estable, ya que se encuentra en la versión 1.4. Esta versión ha sido la utilizada en la elaboración de este proyecto.

3.4.1. Características principales

Algunas de las características que nos ofrece Axis2 son:

Velocidad: Axis2 utiliza su propio modelo de objetos (AXIOM) y StAX (Streaming API for XML). Con ello consigue significativamente más velocidad que en las primeras versiones de Apache Axis.

Poco espacio en memoria: Axis2 ha sido optimizado para funcionar utilizando muy poca memoria.

AXIOM: Axis2 tiene su propio modelo de procesado de mensajes. Además de ser muy potente es muy ligero.

Despliegue de servicios "en caliente": Axis2 tiene la posibilidad de desplegar servicios sin tener que reiniciar el servidor.

Soporte WSDL: Axis2 soporta las versiones 1.1 y 2.0 del Web Service Description Language, que permite de una manera muy sencilla crear los clientes de conexión a los servicios remotos y exportar automáticamente descripciones WSDL de servicios ya desplegados desde Axis2.

Estabilidad: Axis2 dispone de una colección muy estable de interfaces publicados sobre los que se realizan apenas modificaciones.

Flexibilidad: la arquitectura Axis2 da al desarrollador completa libertad para incluir extensiones o modificaciones de la implementación, puesto que el código está disponible.

3.4.2. Participantes de Axis2

En terminología SOAP, un participante es cualquier elemento que tome parte en un servicio Web. También se le conoce como nodo SOAP. La entrega de un único mensaje SOAP está basada en dos participantes, el emisor SOAP y el receptor SOAP. Una única entrega de mensaje es la unidad básica que construye la interacción con los servicios Web.

Cada nodo SOAP puede estar escrito en un lenguaje de programación diferente, como Java, C++, .NET o Perl, pues cada interacción está hecha vía SOAP, que es un mensaje común, entendido por todos los nodos, independientemente del lenguaje en el que estén escritos.

El *middleware* de servicios Web maneja la complejidad de los mensajes SOAP y permite a los desarrolladores trabajar en el lenguaje de programación que deseen. Axis2 maneja el

procesado de mensajes SOAP, junto con otro gran número de funcionalidades, que hacen muy sencilla la labor del desarrollador. A continuación se especifican algunas de ellas:

- Capacidad para desplegar un servicio Web, con o sin WSDL.
- Provee un API de cliente que puede ser utilizado para invocar servicios Web. Este API puede soportar los modelos de envío de mensajes SOAP síncrono y asíncrono.
- Capacidad de enviar y recibir mensajes SOAP mediante diferentes transportes.
- Capacidad de configurar Axis2 y sus componentes a través del despliegue.
- Puede generar los esqueletos de los clientes y servicios Web a partir de un fichero WSDL. De la misma manera, es capaz de generar un fichero WSDL a partir de un servicio Web.

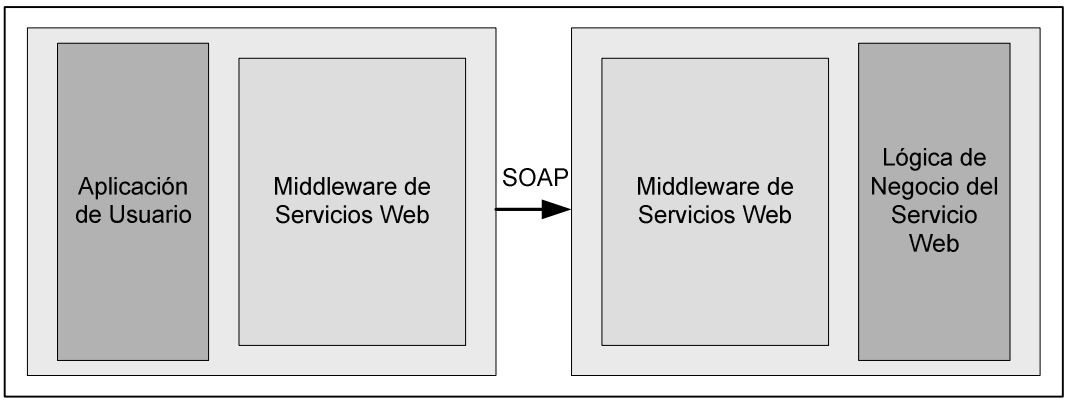


Figura 3-4. Participantes en un escenario Axis2.

Además de las funcionalidades descritas, cabe destacar el buen rendimiento en términos de memoria y velocidad de Axis2. El núcleo de la arquitectura está implementado sobre tres especificaciones principalmente: WSDL, SOAP y WS-Addresing. Otras especificaciones como JAX-RPC, SAAJ y WS-Policy, también tienen cabida en las capas más superficiales de la arquitectura.

3.4.3. Arquitectura

La arquitectura de Axis2 descansa sobre una serie de principios que se encargan de preservar su uniformidad. Son los siguientes:

La arquitectura Axis separa la lógica y los estados. El código que procesa no tiene que tener necesariamente un estado dentro de Axis2. Esto permite ejecutar código libremente por hilos en paralelo.

Toda la información se mantiene en un único modelo de información, lo que permite al sistema ser suspendido y rearrancado.

La arquitectura de Axis2 es modular. Los módulos sobre los que descansa el núcleo son los siguientes:

- Módulo de información. Axis2 define un modelo de información en el que se encuentran todos los estados. El modelo consiste en una jerarquía de información y el sistema gestiona el ciclo de vida de los objetos en esta jerarquía.
- Módulo de procesado XML. El manejo de los mensajes SOAP es la tarea más completa y más importante. La eficiencia de este módulo determina el rendimiento completo. Por ello, tiene sentido delegar esta tarea en un sub-proyecto dentro del proyecto de WebServices de Apache. Éste es AXIOM (Axis Object Model) que encapsula la complejidad de un procesado eficiente de XML.
- Módulo de procesamiento SOAP. Este módulo se encarga de controlar la ejecución del proceso. El modelo define diferentes fases por las que debería pasar la ejecución y el usuario puede heredar este modelo de procesamiento para sus planes específicos.
- Módulo de despliegue. El modelo de despliegue de Axis2 permite al desarrollador desplegar servicios, configurar el transporte y extender el modelo de proceso SOAP para un sistema o servicio concretos.
- Módulo de transporte. Axis2 define un *framework* de transporte flexible. La implementación dispone de una serie de transportes ya desarrollados pero el usuario puede implementar los suyos propios si es necesario.

A continuación se muestan otros módulos:

- Generación de código. Axis2 proporciona una herramienta capaz de generar el código de servidor y de cliente a partir de ficheros descriptores. El código generado simplifica el despliegue del servicio y su invocación desde un cliente, incrementando así la usabilidad de Axis2.

- DataBinding: El *framework* Axis2 Databinding proporciona una manera simple y ligera de compilar un esquema para generar los clientes y servicios Web.

3.4.4. El módulo de información

El módulo de información de Axis2 tiene dos jerarquías principales: contextos y descripciones. La siguiente figura muestra el diagrama UML del modelo:

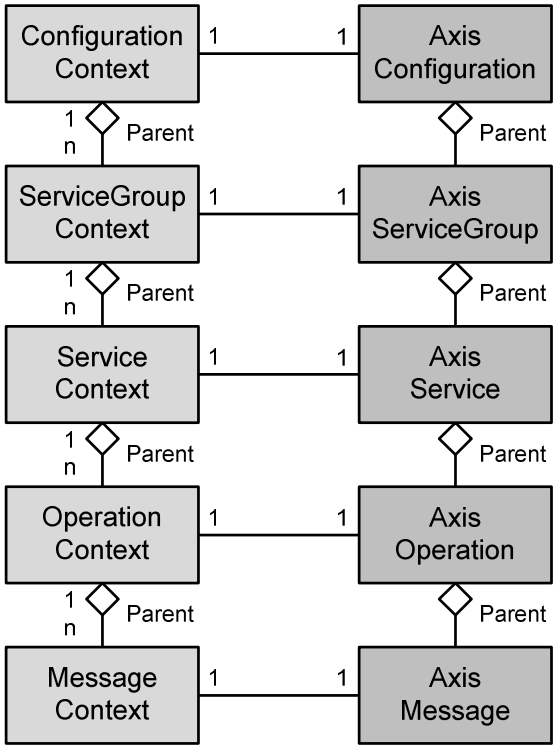


Figura 3-5. Diagrama UML de las jerarquías del módulo de información de Axis2.

Las dos jerarquías están interconectadas, tal y como se puede ver en la **Figura 3-5**. La jerarquía de descripción representa los datos estáticos, que son cargados desde los ficheros de configuración existentes durante el ciclo de vida de Axis2. La jerarquía de contexto maneja toda la información dinámica sobre objetos, que pueden ser instanciados más de una vez, como es el caso de los *MessageContext*.

Estas dos jerarquías crean un modelo que tiene la capacidad de realizar búsquedas mediante clave-valor. Cuando se buscan valores en un nivel correspondiente, el sistema se mueve hacia arriba hasta que encuentra el valor deseado. Esto permite que los niveles inferiores puedan sobreescribir a los niveles superiores. Por ejemplo, cuando un valor se busca en el

MessageContext y no es encontrado, se busca en el *Operation Context* y así sucesivamente hacia arriba en la jerarquía.

Esto permite al usuario declarar y sobreescribir valores, dando como resultado un modelo de configuración muy flexible. Esta flexibilidad puede también perjudicar el sistema, ya que las búsquedas son costosas, especialmente para los parámetros que no existan.

A continuación se explica con más detalle qué hace cada elemento de la jerarquía:

Configuration Context: Maneja el estado en la ejecución de Axis2.

ServiceGroup Context: Maneja información sobre un uso particular en los grupos de servicios. El ciclo de vida de un *ServiceGroup Context* empieza cuando un usuario comienza a interactuar con un servicio que pertenece a un determinado grupo. Esto se puede utilizar para compartir información entre servicios, dentro de un mismo grupo, en una única interacción.

Service Context: El contexto está disponible durante el uso de un determinado servicio. Puede usarse, por ejemplo, para compartir información entre diferentes MEPs (Message Exchange Pattern[3]). El ciclo de vida del contexto depende del ámbito del servicio.

Operation Context: Maneja información sobre la instancia del MEP que se está ejecutando.

Message Context: Maneja información sobre el mensaje que está siendo ejecutado en un determinado momento.

Axis Configuration: Maneja toda la configuración global, como el transporte, módulos globales, parámetros, servicios, etc.

Axis Service Group: Maneja información sobre el tiempo de despliegue de un servicio particular.

Axis Service: Maneja la configuración de las operaciones y del nivel de servicio.

Axis Operation: Maneja la configuración en el nivel de operación.

Axis Message: Maneja la información en el nivel de mensajes, como el esquema de un mensaje concreto.

[3] Un MEP consiste en una plantilla que establece un patrón para el intercambio de mensajes SOAP entre nodos.

3.4.5. El modelo de procesamiento SOAP

La arquitectura de Axis2 identifica dos acciones básicas que el proceso de mensajes SOAP debería realizar: el envío y la recepción de mensajes SOAP. Dicha arquitectura tiene dos flujos para llevar a cabo estas acciones. El *driver* de Axis2 define dos métodos, `send()` y `receive()`, para implementar estas acciones. Estos dos flujos se llaman flujo de entrada y flujo de salida, y los MEP se construyen como una combinación de ambos.

Además, el modelo de procesamiento de Axis2 dispone de unos manejadores que interceptan el mensaje y le proporcionan los servicios que necesite. En general, los manipuladores actúan sobre las cabeceras SOAP y pueden cambiar partes del cuerpo del mensaje si es necesario.

La **Figura 3-6** muestra un esquema del modelo de procesamiento SOAP de Axis2:

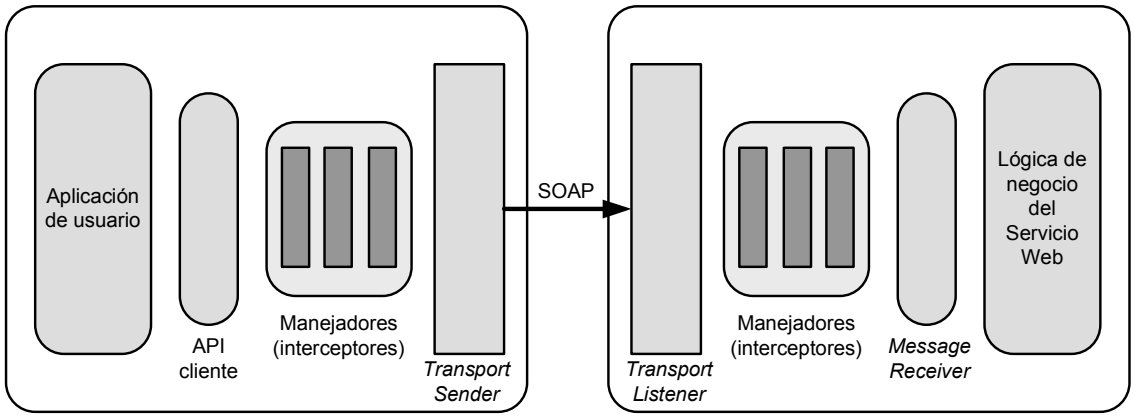


Figura 3-6. Arquitectura del modelo de procesamiento SOAP de Axis2.

Como se puede ver, cuando un mensaje SOAP está siendo enviado a través del cliente, se activa el flujo de salida, que invoca a los manejadores. Finalmente el *Transport Sender*, envía el mensaje al punto de destino. El mensaje SOAP es recibido en el destino por el *Transport Receiver* que lo lee y comienza el flujo de entrada. El flujo de entrada activa los manejadores y termina con el *Message Receiver* que consume el servicio.

Procesado de un mensaje SOAP entrante

Un mensaje SOAP entrante es recogido por el *Transport Receiver*, que está siempre activo a la espera de que lleguen nuevos mensajes. Una vez que el mensaje ha llegado, se *parsean* las cabeceras de trasporte y se crea un *Message Context* a partir del mensaje. Este contexto

encapsula toda la información, incluyendo el mensaje en sí mismo y las cabeceras de transporte, entre otras cosas. Entonces, el flujo de entrada se ejecuta con el contexto del mensaje.

A continuación se podrá ver con detalle lo que ocurre en cada una de las fases de ejecución. Este modelo es aplicable tanto a la parte del servidor como la del cliente.

1. **Transport Phase**: en esta fase los manejadores procesan la información específica de transporte. Validan los mensajes entrantes a través de las cabeceras, añaden datos en los contextos del mensaje, etc.
2. **Pre-Dispatch Phase**: la principal funcionalidad de los manejadores en esta fase es completar el contexto del mensaje para prepararlo para hacer el *dispatching*, por ejemplo, procesando las cabeceras de direcciones del mensaje SOAP y, si es necesario, extrayendo la información y añadiéndola en el contexto del mensaje.
3. **Dispatch Phase**: en esta fase los *dispatchers* buscan la operación y servicio concretos para los que un mensaje SOAP está destinado. Si no encuentra el destino, la ejecución finalizaa devolviendo un error *"Servicio no encontrado"*.
4. **User Defined Phase**: en esta fase actúan los propios manejadores, si los hay.
5. **Message Validation Phase**: una vez que comienza el nivel de ejecución, esta fase valida que el procesado del mensaje se haya hecho correctamente.
6. **Message Processing Phase**: aquí se ejecuta toda la lógica de negocio del mensaje SOAP. El *Message Receiver* asociado a la operación se ejecuta como último manejador de esta fase.

Puede haber más tipos de manejadores en cada una de estas fases, puesto que el usuario puede crear sus propios manejadores que sobreescriban la lógica de proceso en cada una de estas fases.

Procesado de un mensaje SOAP saliente

El flujo de salida es mucho mas simple que el de entrada, ya que la operación a la que hay que redirigir es conocida en el momento en que el flujo es ejecutado. A continuación se muestran las fases que componen el flujo de salida.

1. **Message Initialize Phase**: es la primera fase del flujo de salida. Se utiliza para preparar los manejadores de usuario.

2. **User Phases**: ejecuta las fases de los manejadores definidos por el usuario.
3. **Trasports Phase**: ejecuta cualquier manejador de transporte que haya sido configurado. El último en ejecutarse debería ser el *Transport Sender*, cuya misión consiste en llevar el mensaje SOAP al punto de destino.

3.4.6. Despliegue en Axis2

El modelo de despliegue contiene un mecanismo de configuración de Axis2. Este modelo tiene tres entidades que almacenan esta configuración.

Fichero Axis2.xml

Este fichero contiene la información global del cliente y del servidor. Proporciona la siguiente información:

- Parámetros globales.
- Registra los transportes de entrada y de salida.
- Nombres de las fases definidas por el usuario.
- Módulos comunes a todos los servicios.
- *Message Receivers* definidos de forma global.

Archivo de Servicios

El archivo de servicios, contiene los servicios que va a desplegar en formato *aar* (Axis2 Archive). Dentro del servicio, hay un archivo *META-INF/services.xml* que contiene la siguiente información.

- Parámetros del nivel de servicio.
- Módulos adheridos al nivel de servicio.
- *Message Receivers* específicos para el servicio.
- Operaciones dentro del servicio.

Además, dentro de la carpeta *META-INF* puede haber también clases dependientes y los esquemas utilizados. A continuación se incluye el fichero *services.xml* del servicio *editorial.aar* implementado en este proyexto:

```
<serviceGroup>
  <service name="editorial">
      <messageReceivers>
          <messageReceiver mep=http://www.w3.org/ns/wsdl/in-out
          class="es.pfc.editorial.axis2.ws.EditorialMessageReceiverInOut" />
      </messageReceivers>
      <parameter name="ServiceClass">
          es.pfc.editorial.axis2.ws.EditorialSkeleton
      </parameter>
      <parameter name="useOriginalwsdl">true</parameter>
      <parameter name="modifyUserWSDLPortAddress">true</parameter>
      <operation name="updateStock" mep="http://www.w3.org/ns/wsdl/in-out">
          <actionMapping>
              http://www.example.org/editorial/updateStock
          </actionMapping>
          <outputActionMapping>
              http://www.example.org/editorial/editorial/updateStockResponse
          </outputActionMapping>
      </operation>
  </service>
</serviceGroup>
```

Archivo de Módulos

Cada archivo de módulos debe tener un fichero *META-INF/module.xml*, que contiene los parámetros y las operaciones definidas en el módulo. Además, puede contener las clases dependientes.

3.4.7. Módulo de Transporte

Axis2 dispone de dos constructores para el nivel de transporte, llamados *Transport Senders* y *Transport Receivers*. Se accede a ellos a través de la configuración de Axis2.

El transporte entrante es el que permite a Axis2 recibir el mensaje. El transporte saliente se decide en base a la información de las cabeceras contenidas en el mensaje SOAP (`wsa:ReplyTo` y `wsa:FaultTo`). Si esta información no está disponible y el servidor espera una respuesta, el transporte saliente será el flujo de salida del transporte entrante.

En la parte del cliente, el usuario es libre de especificar el transporte que se utilizará.

Transport Senders y *Transport Receivers* contienen la siguiente información:

- *Transport Sender* para la configuración de salida.
- *Transport Listener* para la configuración de entrada.
- Parámetros de transporte.

Axis2 proporciona soporte a los siguientes transportes:

- **HTTP**: En el transporte HTTP, el *listener* es el *servlet* `org.apache.axis2.transport.http.SimpleHTTPServer` proporcionado por Axis2. El *Transport Sender* utiliza el paquete *commons-httpclient* para conectarse y enviar el mensaje.
- **TCP**: Es más sencillo de utilizar, pero necesita el módulo *WS-Addresing* para ser funcional.
- **SMTP**: Trabaja con una única cuenta de correo. El *Transport Receiver* consiste en un hilo que comprueba la llegada de *e-mails* a intervalos fijos de tiempo.
- **JMS**: Java Messaging Service. Sistema de mensajería estándar de Java.

4. La base de datos

4.1. Introducción

Todas las aplicaciones Web necesitan de un sistema de almacenamiento persistente para guardar los datos y permitir su modificación y acceso en un futuro. Este almacenamiento puede ser de varios tipos, pero lo más frecuente es emplear una base de datos relacional.

Dentro del conjunto de bases de datos relacionales que existen, en el desarrollo de este proyecto se ha usado MySQL. A continuación se muestran sus características principales y las razones para su empleo.

4.2. MySQL

MySQL es un sistema de gestión de base de datos relacional (RDBMS, por sus siglas en inglés), multihilo y multiusuario. MySQL AB desarrolla MySQL como *software* libre en un esquema de licenciamiento dual. Por un lado lo ofrece bajo la licencia GNU GPL y por otro, lo vende a empresas que deseen incorporarlo en productos privados con una licencia que les permita ese uso.

Al contrario de lo que ocurre en proyectos como Apache, donde el *software* se desarrolla por una comunidad pública y el *copyright* del código pertenece al autor individual, MySQL está patrocinado por una empresa privada, que posee el *copyright* de la mayor parte del código. Esto es lo que permite el sistema de licenciamiento dual mencionado anteriormente. Además de la venta de licencias privadas, MySQL AB también ofrece soporte y servicios.

Una base de datos relacional almacena datos en tablas separadas, en lugar de colocar todos los datos en un gran almacén. Esto añade velocidad y flexibilidad. El lenguaje de acceso a los datos está estandarizado y se conoce como SQL (Structured Query Language). Cada conjunto de bases de datos está gestionado por el servidor de bases de datos.

4.2.1. Especificaciones

Algunas de las características del *software* de base de datos MySQL son las siguientes:

Interioridades y portabilidad

- MySQL está escrito en C y C++ y ha sido probado con un amplio rango de compiladores diferentes.
- Funciona en multitud de plataformas, como Windows, Linux, Mac OS, FreeBSD, Amiga o HP-UX, entre otras.
- Hay APIs disponibles para C, C++, Eiffel, Java, Perl, PHP, Python, Ruby y TCL.
- Usa GNU Automake, Autoconf y Libtool para la portabilidad.
- Comportamiento multihilo mediante el empleo de hilos del *kernel*. Pueden usarse fácilmente múltiples CPU si están disponibles.
- Proporciona sistemas de almacenamiento transaccionales y no transaccionales.
- Usa tablas en disco de tipo B-Tree (MyISAM) muy rápidas, con compresión de índice.
- Tiene un sistema de reserva de memoria muy rápido basado en hilos.
- Los `JOINS` (consultas cuyo resultado se obtiene de la unión de varias tablas) son muy rápidos porque utiliza un *multi-join* de un paso optimizado.
- Usa tablas *hash* en memoria que son usadas como tablas temporales.
- Las funciones SQL están implementadas usando una biblioteca altamente optimizada, de forma que sean tan rápidas como permita el sistema. Normalmente no hay reserva de memoria tras toda la inicialización para consultas.
- El código MySQL se prueba con Purify, un detector de memoria perdida comercial, así como con Valgrind, una herramienta GPL.
- El servidor de base de datos de MySQL está disponible como un programa separado para usar en un entorno de red cliente/servidor. También está disponible como biblioteca y puede ser embebido en aplicaciones autónomas. Dichas aplicaciones pueden usarse por sí mismas o en entornos donde no hay red disponible.

Tipos de columnas

- En MySQL existe un gran número de tipos de datos: enteros con y sin signo de 1, 2, 3, 4, y 8 *bytes* de longitud, `FLOAT`, `DOUBLE`, `CHAR`, `VARCHAR`, `TEXT`, `BLOB`, `DATE`, `TIME`, `DATETIME`, `TIMESTAMP`, `YEAR`, `SET`, `ENUM` y tipos espaciales OpenGIS.
- Los registros pueden ser de longitud fija o variable.

Sentencias y Funciones

MySQL da un soporte completo para operadores y funciones en las cláusulas de consultas `SELECT` y `WHERE`.

Dispone de soporte completo para las cláusulas SQL `GROUP BY` y `ORDER BY`. Soporte de funciones de agrupación como `COUNT()`, `COUNT (DISTINCT …)`, `AVG()`, `STD()`, `SUM()`, `MAX()`, `MIN()` y `GROUP CONCAT()`.

Permite las consultas `LEFT OUTER JOIN` y `RIGHT OUTER JOIN` cumpliendo estándares de sintaxis SQL y ODBC.

Permite tener *alias* en tablas y columnas tal y como lo requiere el estándar SQL.

Las operaciones `DELETE`, `INSERT`, `REPLACE` y `UPDATE` devuelven el número de filas que han sido afectadas con su uso. Es posible devolver el número de filas que serían afectadas usando un *flag* al conectar con el servidor.

El comando específico de MySQL `SHOW` puede usarse para obtener información acerca de la base de datos, el motor de base de datos, las tablas y los índices. El comando `EXPLAIN` puede usarse para determinar cómo el optimizador resuelve una consulta.

Los nombres de funciones no colisionan con los nombres de tabla o columna. Por ejemplo, *ABS* es un nombre válido de columna. Como única restricción se tiene que para una llamada a una función no se permiten espacios entre el nombre de función y el paréntesis que debe llevar a continuación.

En MySQL se pueden mezclar tablas de distintas bases de datos en la misma consulta.

Seguridad

Utiliza un sistema de privilegios y contraseñas muy flexible y seguro, que además permite verificación basada en el *host*. Las contraseñas son seguras porque todo el tráfico de contraseñas está encriptado cuando se conecta con un servidor.

Escalabilidad y límites

MySQL da soporte a grandes bases de datos. MySQL Server puede albergar bases de datos que contienen alrededor de 50 millones de registros.

Se permiten hasta 64 índices por tabla. Cada índice puede contener de 1 a 16 columnas o partes de columna. El máximo ancho de límite son 1000 *bytes*. Un índice puede usar prefijos de una columna para los tipos de columna `CHAR`, `VARCHAR`, `BLOB` y `TEXT`.

Conectividad

Los clientes pueden conectar con el servidor MySQL usando *sockets* TCP/IP en cualquier plataforma. En sistemas Windows de la familia NT (NT, 2000, XP o 2003), los clientes pueden usar *named pipes* para la conexión. En sistemas Unix, los clientes pueden conectar usando ficheros *socket* Unix.

En MySQL 5.0, los servidores Windows soportan conexiones con memoria compartida si se inicializan con la opción `--shared memory`. Los clientes se pueden conectar a través de memoria compartida usando la opción `--protocol=memory`.

La interfaz para el conector ODBC (MyODBC) proporciona a MySQL soporte para programas clientes que usen conexiones ODBC (Open Database Connectivity). Por ejemplo, puede usar Microsoft Access para conectar al servidor MySQL. Los clientes pueden ejecutarse en Windows o Unix. El código fuente de MyODBC está disponible.

La interfaz para el conector J MySQL proporciona soporte para clientes Java que usen conexiones JDBC. Estos clientes pueden ejecutarse en Windows o Unix. El código fuente para el conector J también está disponible.

Internacionalización

El servidor puede proporcionar mensajes de error a los clientes en multitud de idiomas.

Dispone de un soporte completo para distintos conjuntos de caracteres, incluyendo latin1 (ISO-8859-1), german, big5, ujis, etc. También existe soporte para Unicode.

Todos los datos se guardan en el conjunto de caracteres elegido. Todas las comparaciones para columnas normales de cadenas de caracteres son *case-insensitive*.

La ordenación se realiza de acuerdo con el conjunto de caracteres elegido. MySQL Server soporta diferentes conjuntos de caracteres, que deben ser especificados en tiempo de compilación y de ejecución.

Clientes y herramientas

MySQL Server tiene soporte para órdenes SQL para *chequear*, optimizar y reparar tablas. Estos comandos están disponibles a través de la línea de comandos y el cliente *mysqlcheck.* MySQL también incluye *myisamchk,* una utilidad de línea de comandos muy rápida para efectuar estas operaciones en tablas MyISAM.

Todos los programas MySQL pueden invocarse con las opciones `--help` o -? para obtener asistencia en línea.

4.2.2. Las tablas en MySQL: MyISAM

A partir de la versión 3.23 de MySQL, las tablas denominadas de método de acceso secuencial indexado, ISAM, fueron sustituidas por otro tipo de tablas, MyISAM. La principal diferencia entre ambas es que el índice de las tablas MyISAM es mucho más pequeño que el de las ISAM, de manera que una consulta `SELECT` con un índice sobre una tabla MyISAM utilizará menos recursos del sistema. A continuación se describen algunos tipos de tablas en MySQL:

Tablas estáticas

Las tablas estáticas tienen un número de registros fijo. Cada registro tiene asignado exactamente 10 *bytes*. Este tipo de tablas se caracterizan por ser muy rápidas, aunque requieren más espacio en disco. Resultan sencillas de reconstruir tras un fallo y de almacenar en la *caché*.

Tablas dinámicas

Las columnas de las tablas dinámicas tienen diferentes tamaños. Aunque este tipo de dato ahorra espacio respecto a las tablas estáticas, su uso resulta sin embargo más complejo.

Las tablas de tipo dinámico presentan las siguientes características:

- Todas las columnas de cadena son dinámicas.
- Por regla general, ocupan mucho menos espacio de disco que las tablas fijas.
- Las tablas requieren un mantenimiento regular para evitar su fragmentación.

- No resultan tan sencillas de reconstruir tras un fallo como las estáticas, especialmente si las tablas están muy fragmentadas.

Tablas comprimidas

Las tablas comprimidas son tablas cuyos datos no se pueden modificar, sólo leer. Utilizan mucho menos espacio en disco que otras tablas. Son ideales para su uso con datos comprimidos que no cambien, ya que solo se pueden leer y no escribir, y para entornos en los que el espacio de almacenamiento sea escaso.

Las tablas comprimidas presentan las siguientes características:

- Son tablas son mucho más pequeñas.
- Como cada registro se comprime de forma separada, la carga de acceso es reducida.
- Cada columna se puede comprimir de forma diferente, utilizando distintos algoritmos de compresión.
- Tanto las tablas estáticas como las dinámicas pueden convertirse en tablas comprimidas.

Tablas MERGE

Las tablas MERGE se crean a partir de la fusión de tablas MyISAM iguales. Por lo general se usan cuando las tablas MyISAM comienzan a resultar demasiado grandes.

Entre las ventajas de estas tablas se pueden mencionar las siguientes:

- Resultan más rápidas en determinadas situaciones.
- El tamaño de la tabla es más pequeño.

Desventajas de las tablas MERGE:

- Resultan mucho más lentas en búsquedas.
- El comando `REPLACE` no funciona sobre ellas.

Tablas MEMORY

Las tablas MEMORY o HEAP son el tipo de tabla más rápido porque se almacena en memoria RAM y utilizan un índice *hash* asignado. Como se almacenan en memoria volátil, en el caso de un fallo del sistema los datos se pierden. Las tablas, por sí mismas, continúan

existiendo, ya que sus definiciones se almacenan en fichero de tipo *.frm* en el disco, pero están vacías cuando se reinicia el servidor.

Tablas INNODB

Las tablas INNODB son tablas de transacción segura, lo que implica que disponen de las funciones `COMMIT` y `ROLLBACK`. En una tabla MyISAM, la tabla entera se bloquea al realizar funciones de inserción. Durante esa fracción de segundo no se puede ejecutar ninguna otra instrucción sobre la tabla. INNODB utiliza funciones de bloqueo a nivel de fila de manera que solo se bloquee dicha fila y no toda la tabla, y se puedan seguir aplicando instrucciones sobre otras filas.

4.2.3. Estabilidad de MySQL

Al tratarse MySQL Server de un *software* de gestión de base de datos libre y gratuito, suele preocupar a los desarrolladores su estabilidad.

Cuando se va a desarrollar una aplicación, especialmente si se trabaja con grandes conjuntos de datos o un número importante de conexiones concurrentes, es necesario preguntarse en primer lugar si el sistema elegido soportará con amplitud los niveles de carga a los que va a ser sometido.

Cuando MySQL fue distribuido entre un público más amplio, los nuevos usuarios encontraron partes de código no probados con sus consecuentes fallos. Cada nueva versión lanzada ha servido para mejorar el sistema y ha subsanado los fallos encontrados. Estas nuevas versiones han tenido muy pocos problemas de portabilidad, incluso considerando que cada una de ellas ha aumentado las funcionalidades respecto de la anterior.

Con esto, se puede considerar a MySQL como un sistema suficientemente estable y válido para el desarrollo de aplicaciones comerciales.

4.3. Ventajas en el uso de MySQL

Se ha seleccionado MySQL para el desarrollo de este proyecto por muchas de sus ventajas. A continuación se indican algunas de ellas.

- Es gratuita, por lo que está al alcance de cualquier desarrollador, no sólo para proyectos profesionales.

- Su uso es muy sencillo tanto desde las herramientas gráficas como desde la interfaz de línea de órdenes.
- Escalabilidad, ya que permite manejar bases de datos muy grandes.
- MySQL dispone de APIs para múltiples plataformas diferentes.
- Conectividad, permite conexiones entre diferentes máquinas con distintos sistemas operativos. Es muy usual en el desarrollo de aplicaciones profesionales que servidores Linux o Unix sirvan datos a ordenadores con Windows, Linux, Solaris, etc.
- Es multihilo, por lo que puede beneficiarse de sistemas multiprocesador.
- Para acceder a los datos se utiliza el lenguaje SQL, que es un lenguaje estándar, por lo que las consultas hechas usando este lenguaje son fácilmente portables a otros sistemas y plataformas.
- El motor de base de datos de MySQL también dota de seguridad en el acceso a las bases de datos. Esto lo hace en forma de permisos y privilegios, determinados usuarios disponen de permisos para consulta o modificación de determinadas tablas. Esto permite compartir datos sin que peligre la integridad de la base de datos o protegiendo determinados contenidos.

4.4. Inconvenientes en el uso de MySQL

Como cualquier herramienta, MySQL también tiene sus inconvenientes. A continuación se muestran algunas de las desventajas en el uso de MySQL como sistema gestor de base de datos.

- La mayoría de tipos de tabla no soporta *rollbacks*, con lo que no se puede deshacer una operación sobre una tabla de la base de datos. Tampoco soporta *subselects* ni transacciones.
- MySQL no gestiona la integridad referencial, dejándola en manos del programador de la aplicación.
- MySQL es una base de datos muy rápida en la lectura cuando utiliza el motor no transaccional MyISAM, pero puede provocar problemas de integridad en entornos de alta concurrencia en la modificación. En aplicaciones Web hay baja concurrencia

en la modificación de datos y en cambio el entorno es intensivo en lectura de datos, lo que hace a MySQL ideal para este tipo de aplicaciones.

5. Otras tecnologías empleadas

5.1. El servidor de aplicaciones: Apache Tomcat

El servidor Apache Tomcat es un contenedor de *servlets* con un entorno JSP, esto es, un interfaz de ejecución que permite al usuario manejar e invocar *servlets*. Incluye el compilador Jasper que compila las páginas jsp convirtiéndolas en *servlets*.

Tomcat puede funcionar como un servidor Web por sí mismo. En sus inicios existió la idea de que el uso de Tomcat de forma autónoma era sólo recomendable para entornos de desarrollo y entornos con requisitos mínimos de velocidad y gestión de transacciones. Hoy en día ya no existe esa percepción y Tomcat se usa como servidor web autónomo en entornos con alto nivel de tráfico y alta disponibilidad.

Tomcat es mantenido y desarrollado bajo el proyecto Jakarta de la Apache Software Foundation. Los usuarios disponen de libre acceso a su código fuente y a los binarios ejecutables en los términos establecidos en la Apache Software Licence. Las primeras distribuciones de Tomcat fueron las versiones 3.0.x. Las versiones más recientes son las 6.x, que implementan las especificaciones de Servlet 2.5 y de JSP 2.1. A partir de la versión 4.0 Tomcat utiliza el contenedor de *servlets* Catalina.

5.1.1. Arranque y parada de Tomcat

Para arrancar y parar el servidor, existen *scripts* que realizan dichas tareas. Estos *scripts* se encuentran en el directorio *bin*, dentro de la carpeta en la que se instala Tomcat, y están disponibles para ser ejecutados en plataformas Unix y Win32.

Para Unix:

- *bin/startup.sh*. *Shell script* para arranque del servicio que ejecuta Tomcat.
- *bin/shutdown.sh*. *Script* para la parada del servicio.

Para plataformas Win32:

- *bin\startup.bat*. Archivo ejecutable que arranca la aplicación.

- *bin\shutdown.bat*. Detiene Tomcat.

5.1.2. La estructura de directorios de Tomcat

Dentro del directorio en el que se instala Tomcat se crean una serie de directorios que se indican a continuación.

bin: arranque y parada del servidor y otros *scripts* ejecutables.

conf: ficheros XML y DTD necesarios para configuración de Tomcat.

lib: conjunto de bibliotecas utilizadas por Tomcat y las aplicaciones Web.

logs: archivo de registro de Apache Tomcat.

temp: almacena archivos temporales.

webapps: directorio que contiene las aplicaciones Web.

work: almacenamiento temporal de ficheros y directorios.

5.1.3. Los ficheros de configuración

Los ficheros de configuración de Tomcat se encuentran dentro de la carpeta *conf*. La configuración del servidor de aplicaciones está gestionada principalmente por dos ficheros escritos en XML. Estos ficheros son *server.xml* y *web.xml*, y contienen las directivas de configuración del servidor de aplicaciones. En el fichero *context.xml* se definen las directivas referentes a las aplicaciones Web.

El fichero server.xml

Es el archivo de configuración principal de Tomcat. Tiene dos objetivos:

- Proporciona la configuración inicial para los componentes de Tomcat.
- Especifica la estructura de Tomcat. Contiene los componentes que permiten que Tomcat arranque y se contruya a sí mismo.

Algunos de los elementos que se definen en este fichero son los siguientes:

Server: Es el elemento superior del fichero *server.xml* que define un servidor Tomcat. En general no es necesario modificarlo. Un elemento `Server` puede contener elementos `Service` y `GlobalNamingResources`.

GlobalNamingResources: define los elementos JNDI del servidor.

Service: Representa el conjunto de uno o más elementos de tipo `Connector` asociados al servicio Catalina, encargado de procesar las peticiones entrantes. Todas las implementaciones del servicio aceptan los atributos *className,* que identifica la clase de la implementación a utilizar, y *name,* que corresponde al nombre del servicio.

Connector: El objeto `Connector` es el responsable del control de los hilos de ejecución en Tomcat y de leer y escribir las peticiones y respuestas desde los *sockets* conectados a los distintos clientes. La configuración de los conectores incluye información como:

- El protocolo de comunicación del conector. Puede ser HTTP o AJP.
- El puerto TCP/IP donde escucha el controlador.

El fichero web.xml

Este fichero define los valores por defecto de todas las aplicaciones Web cargadas dentro del servidor Tomcat. Por cada aplicación desplegada se procesa este fichero, seguido del propio *web.xml* de cada aplicación.

Mediante este archivo se configuran, entre otras cosas, recursos que pueden ser utilizados por todas las aplicaciones. En él no se deben configurar recursos propios de una única aplicación puesto que ello se debería hacer en el *web.xml* de la aplicación concreta.

Además, en este fichero también se configura el *servlet* por defecto al que cada aplicación invocará en última instancia. Los *servlets* propios de una aplicación concreta se configuran en el *web.xml* de la aplicación.

También aquí se definen los tipos MIME y el *welcome-file*, que es el primer fichero que se busca cuando se carga el contexto de la aplicación Web si no se especifica ninguno concreto.

El fichero context.xml

En este fichero se define la información de contexto cargada por las aplicaciones Web alojadas en el servidor Tomcat. Cada aplicación está referenciada por un elemento `Context`.

Context: Cada `Context` representa una aplicación Web funcionando dentro de Tomcat. Se pueden definir tantos elementos `Context` como sea necesario, siempre que cada uno tenga un *context path* único.

Tomcat necesita la siguiente información para definirse completamente:

- El *path* donde se localiza el contexto. Este puede ser un *path* completo o relativo al directorio raíz del `ContextManager`. Debe ser único dentro de un mismo servidor. Si se especifica *""* como *path* de contexto, se está haciendo referencia a la aplicación Web por defecto, que procesará todo lo que no esté asignado a otros contextos.
- Directorio físico dónde se encuentra la aplicación.
- Nivel de depuración usado para los mensajes de registro.
- Un atributo que indica si es *reloadable*, esto es, que se permita aplicar cambios sin tener que reiniciar el servidor.

Dentro de este elemento se pueden anidar una serie de elementos de utilidad. Algunos de ellos se definen a continuación:

Resources: configura el gestor de recursos empleado para el acceso a los recursos estáticos asociados a una aplicación Web. Aquí se puede configurar, por ejemplo, el gestor de conexiones a la base de datos.

Loader: permite configurar un *class loader* específico para la aplicación. Normalmente la configuración por defecto suele ser suficiente.

5.1.4. Manager de Tomcat

El *manager* de Tomcat es una interfaz Web de administración. Tras la instalación de Tomcat, y por razones de seguridad, no se puede acceder al *manager* hasta que se haya creado un usuario de Tomcat con rol de administrador. Para ello se debe editar el fichero *tomcat-users.xml* que se encuentra dentro del directorio *conf* y añadirle la siguiente directiva:

```
<role rolename="manager"/>
<user username="admin" password="xxxxxxx" roles="manager"/>
```

Una vez creado el usuario, se reinicia Tomcat y se puede comenzar a usar el *manager*. Esta interfaz consta de cinco partes:

Message: Muestra el resultado de las órdenes que se han enviado al *manager*. Puede ser OK o FAILED.

Manager: Muestra distintas opciones, como recargar la lista de aplicaciones instaladas, acceder a la documentación o ver el estado del motor de Tomcat.

Applications: Muestra la lista de aplicaciones que está ejecutando Tomcat y su estado. Además se pueden parar (*stop*), iniciar (*start*), recargar (*reload*) o borrar (*undeploy*).

Deploy: Permite desplegar aplicaciones directamente desde esta interfaz.

Server Information: Contiene información del servidor.

5.1.5. DataSource gestionado por el contenedor

Cuando una aplicación accede a una base de datos es necesario obtener una conexión a dicha base de datos, lo cual es un proceso muy costoso que puede ocasionar bloqueos e incluso la caída del servidor. Cuando se trata de una aplicación en la que los accesos pueden ser masivos, como es el caso de la tienda electrónica, la gestión de estas conexiones ha de estar controlada. Esto se hace con un *Data Source* o *pool de conexiones*. El gestor de este *pool* de conexiones crea y reserva en memoria un número determinado de conexiones a la base de datos y las facilita a la aplicación cuando ésta la necesita. La aplicación realiza las consultas necesarias y devuelve la conexión al gestor del *pool* de conexiones.

Esta gestión de las conexiones se puede realizar de diversas maneras, una de ellas es la que ofrece el servidor de aplicaciones Tomcat. Mediante los ficheros de configuración mencionados anteriormente se puede configurar un *pool* de conexiones que estará gestionado directamente por el contenedor. Las directivas que se han de incluir son las siguientes:

En el fichero *server.xml*:

```
<Resource name="jdbc/mySql"
  auth="Container"
  type="javax.sql.DataSource"
  driverClassName="com.mysql.jdbc.Driver"
  url="jdbc:mysql://localhost:3306/elephant" username="root"
  password="root"
  maxActive="10"
  maxIdle="10"
  maxWait="-1" />
```

En el fichero *web.xml*:

```
<resource-ref>
  <description>MySQL Datasource</description>
  <res-ref-name>jdbc/mySql </res-ref-name>
  <res-type>javax.sql.DataSource</res-type>
  <res-auth>Container</res-auth>
</resource-ref>
```

Con estos dos sencillos pasos se define el *pool* de conexiones y se puede utilizar en la aplicación.

5.2. Hojas de estilo en cascada: CSS

Las hojas de estilo en cascada, *Cascading Style Sheets*, son un mecanismo simple que describe cómo va a ser la presentación de un documento escrito en HTML o XML.

CSS se utiliza para dar estilo a documentos HTML y XML separando completamente el contenido de la presentación. Estas hojas de estilo definen la forma de mostrar las etiquetas HTML y XML. CSS permite a los desarrolladores controlar el estilo y el formato de múltiples páginas Web al mismo tiempo. Cualquier cambio en el estilo marcado para un elemento en la CSS afectará a todas las páginas vinculadas a ella en las que aparezca ese elemento.

5.2.1. Funcionamiento

CSS funciona en base a reglas, es decir, declaraciones sobre el estilo de uno o más elementos. Las hojas de estilo están compuestas por una o más de esas reglas aplicadas a un documento HTML o XML. La regla tiene dos partes: un selector y su declaración, que a su vez está compuesta por una propiedad y el valor que se le asigne.

El selector hace referencia al componente sobre el que se va a aplicar la propiedad. Funciona como enlace entre el documento y el estilo, especificando los elementos que se van a ver afectados por esa declaración. Los elementos que se pueden utilizar son tanto etiquetas HTML estándar como bloques referenciados por un identificador propio.

5.2.2. Formas de dar estilo a un documento

Hay diversas formas de dar estilo a un documento, algunas de las cuales se muestran a continuación.

Hojas de estilo externas al documento

Una hoja de estilo externa no es más que un documento de texto en el que están definidos todos los estilos y los selectores sobre los que se aplicará.

Esta hoja de estilo se asocia a los documentos HTML mediante la directiva indicada en negrita como ejemplo a continuación.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN">
<html>
  <head>
    <title>Título</title>
    <link rel="stylesheet" type="text/css" href="css/style.css" />
  </head>
  <body>
  .
  .
  .
  .
  </body>
</html>
```

Esta opción facilita el mantenimiento de las hojas de estilo y la posibilidad de utilizar *skins* en la aplicación. Esto implica que cambiando únicamente las hojas de estilo que se van a utilizar puede cambiar por completo el *Look & Feel* de una aplicación.

Hojas de estilo internas al documento

Si no se van a tener hojas de estilo muy grandes también está la posibilidad de colocar el estilo dentro del propio documento. De esta forma los estilos serán reconocidos antes de que la página se cargue por completo.

Esta opción es más recomendable si el estilo va a afectar a una sola página. En el caso de aplicaciones grandes, donde un mismo estilo afecta a varias páginas, se recomienda la primera opción, ya que, como se ha indicado, facilita el mantenimiento y evita la duplicación de código. De nuevo, se muestra un ejemplo para HTML.

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN">
<html>
  <head>
    <title>hoja de estilo interna</title>
    <style type="text/css">

      body {
        padding-left: 11em;
        font-family: Georgia, "Times New Roman", serif;
        color: red;
        background-color: #d8da3d;
      }

      h1 {
        font-family: Helvetica, Geneva, Arial, sans-serif;
      }

    </style>
  </head>
```

```
  <body>
    <h1>Aqui se aplica el estilo de letra para el Titulo</h1>
  </body>
</html>
```

Estilos aplicados sobre el elemento

También se puede aplicar el estilo directamente sobre aquellos elementos que lo permitan a través del atributo `<style>` dentro de los componentes del `<body>`. Esta forma de emplear CSS es quizá la menos recomendable ya que pierde algunas de las ventajas que dan las hojas de estilo al mezclarse el contenido con la presentación.

5.2.3. Ventajas en el uso de CSS

Una de las ventajas en el uso de CSS es que facilita el mantenimiento de los estilos de las páginas Web, al encontrarse todo el estilo en un mismo repositorio. Además, se elimina la duplicación de código, ya que un mismo estilo se puede aplicar a todos los elementos de una aplicación.

También las hojas de estilo permiten que una aplicación pueda ser vista de diferente forma en diferentes dispositivos. Esto es, se podrá definir cómo se visualizará una misma aplicación en, por ejemplo, un PC o en un PDA.

Los navegadores permiten a los usuarios especificar una hoja de estilo concreta a aplicar a cualquier sitio Web, con lo que aumenta considerablemente la accesibilidad. Por ejemplo, personas con deficiencias visuales pueden configurar su propia hoja de estilo local para aumentar el tamaño del texto o remarcar más los enlaces.

Utilizar CSS ayuda a que el código de una página Web sea más limpio y claro a los ojos de un motor de búsqueda.

CSS es una tecnología muy potente y permite maquetar un documento de un modo mucho más exacto a como lo permite HTML.

5.2.4. Inconvenientes en el uso de CSS

Aunque las ventajas en el uso de CSS son muchas también existen algunos inconvenientes. El más importante es el soporte irregular que tienen las CSS por parte de los navegadores. Ciertas propiedades que funcionan en unos navegadores no lo hacen en otros, o existen diferencias dentro de un mismo navegador en función del sistema operativo sobre el que se

utilice. También existen diferencias entre diferentes versiones de un mismo navegador. Esto puede provocar que algunas de las propiedades de las CSS no sean reconocidas por el navegador, lo que provoca que el aspecto de la página no sea el deseado.

Las CSS son una herramienta que puede resultar muy efectiva para lograr una presentación atractiva de la página siempre que ésta no sea completamente dependiente de la hoja de estilo. Se deben considerar en todo momento aquellos navegadores que no soportan CSS, cuidando que los mismos puedan mostrar la página correctamente y en su totalidad aún cuando nuestras reglas de estilo no sean aplicadas.

5.3. Los registros de log: Log4j

Para todas las aplicaciones es muy importante almacenar registros de *log*. Estos registros son el punto de partida para poder solucionar cualquier tipo de problema específico de una aplicación o conocer el comportamiento de la misma.

Log4j es una biblioteca *OpenSource* desarrollada por la Apache Software Foundation que permite a los desarrolladores elegir la salida y el nivel de granularidad de los mensajes o *logs* en tiempo de ejecución mediante el uso de archivos de configuración externos.

Log4J también ha sido implementado para otros lenguajes además de Java, como C, C++, C#, Perl, Python, Ruby o Eiffel.

5.3.1. Loggers, Appenders y Layouts

Log4j tiene tres componentes principales, *loggers*, *appenders* y *layouts*. Estos tres componentes trabajan juntos para permitir a los desarrolladores generar mensajes dependiendo de las acciones y niveles de evento, y controlar en tiempo de ejecución el formato y almacenamiento de estos mensajes.

Logger

Es la clase principal de Log4j. Permite definir entidades, categorías o espacios de *log*. La clase *logger* sigue una jerarquía a la hora de definir categorías que es análoga a la definición de los paquetes de Java. Siempre hay una categoría raíz (*root logger*).

A cada uno de los *loggers* se les puede asignar una prioridad o nivel diferente. Por defecto Log4j tiene cinco niveles de prioridad de los mensajes (DEBUG, INFO, WARN, ERROR y FATAL), además de otros dos niveles extras (ALL y OFF).

- **DEBUG**: se utiliza para escribir mensajes de depuración, este *log* no debe estar activado cuando la aplicación se encuentre en producción.
- **INFO**: se utiliza para mensajes similares al modo *verbose* en otras aplicaciones.
- **WARN**: se utiliza para mensajes de alerta sobre eventos de los que se desea mantener constancia, pero que no afectan el correcto funcionamiento del programa.
- **ERROR**: se utiliza en mensajes de error de la aplicación que se desean guardar, ante eventos que afectan al programa pero no interrumpen su ejecución, como por ejemplo que algún parámetro de configuración no es correcto y se carga el parámetro por defecto.
- **FATAL**: se utiliza para mensajes críticos del sistema. Generalmente después de guardar el mensaje el programa finalizará.
- **ALL**: este es el nivel más bajo posible, habilita todos los *logs*.
- **OFF**: este es el nivel más alto posible, deshabilita todos los *logs*.

El comportamiento de los *loggers* es jerárquico, tal y como se muestra en la tabla siguiente:

Nivel de logger	**Se mostrarán los mensajes de nivel**				
	debug	info	warn	Error	fatal
DEBUG					
INFO	X				
WARN	X	X			
ERROR	X	X	X		
FATAL	X	X	X	X	
ALL					
OFF	X	X	X	X	X

Un *logger* mostrará únicamente los mensajes de un nivel igual o superior a él. Si el nivel de un *logger* no está definido heredará el nivel de su *logger* padre. Si no tiene un padre definido heredará el nivel del *root logger*, que siempre está disponible y que tiene por defecto asignado el nivel DEBUG.

Una clase generará solo los mensajes asociados a la prioridad activa en ese momento o a mensajes con prioridad mayor.

Appender

Appender es la clase utilizada para definir las salidas de *log*. Las posibilidades de salida son múltiples:

- **ConsoleAppender**: salida por consola. No se suele utilizar, especialmente en entornos de producción.
- **FileAppender**: a un fichero de texto.
- **RollingFileAppender**: versión del *FileAppender* al que se limita el tamaño. Permite copias de respaldo.
- **DailyRollingFileAppender**: versión del *FileAppender* que se reinicia cuando ha pasado un tiempo determinado. Se suelen reiniciar diariamente.
- **SMTPAppender**: la salida se envía por correo electrónico.
- **SocketAppender**: la salida va a un servidor remoto.
- **JDBCAppender**: a una base de datos.
- **JMSAppender**: la salida se obtiene como un mensaje JMS (Java Message Service) de tipo *ObjectMessage*.
- **NullAppender**: la salida no se envía a ningún dispositivo.
- **AsyncAppender:** permite a los usuarios almacenar *logs* de forma asíncrona. Se pueden añadir múltipes *appender* a un único *AsyncAppender*.

Además de estos, hay otro gran número de *appenders* disponibles, y el desarrollador puede implementar el que se ajuste a sus necesidades extendiendo uno de los disponibles.

Layout

Los *Layout* se encargan de dar formato a los mensajes de salida. Permiten presentar el mensaje con el formato necesario para almacenarlo en un fichero de texto (*SimpleLayout* y *PatternLayout*), en una tabla HTML (*HTMLLayout*) y en un archivo XML (*XMLLayout*). También se puede añadir información adicional al mensaje, como la fecha en que se generó, la clase que lo generó, el nivel que posee, etc.

5.3.2. Configuración

El API de Log4j es totalmente configurable. Se puede configurar de dos maneras, mediante un archivo XML o mediante un fichero en formato Java Properties. En este fichero de *properties*, normalmente llamado *log4j.properties*, se definen los parámetros de configuración de la forma `clave=valor`.

A continuación se muestra como ejemplo un fichero de *properties*:

```
#################################################################
### Configuración para LOCAL                                  ###
#################################################################
#log4j.rootCategory=DEBUG, LOGFILE, CONSOLE

#log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender

#log4j.appender.CONSOLE=org.apache.log4j.ConsoleAppender
#log4j.appender.CONSOLE.layout=org.apache.log4j.PatternLayout
#log4j.appender.CONSOLE.layout.ConversionPattern=%-5p %c %x - %m%n


#################################################################
### Configuración para DESARROLLO, PREPRODUCCION, PRODUCCION ###
###   Sólo nos interesa el nivel de ERROR                     ###
###   No hay salida de consola                                ###
#################################################################
log4j.rootCategory=ERROR, diario


#################################################################
### Configuración Común                                       ###
#################################################################
log4j.appender.LOGFILE=org.apache.log4j.DailyRollingFileAppender
log4j.appender.LOGFILE.file=${catalina.base}/logs/aplicacion.log
log4j.appender.LOGFILE.append=true
log4j.appender.LOGFILE.DatePattern='.'yyyy-MM-dd

log4j.appender.LOGFILE.layout=org.apache.log4j.PatternLayout
log4j.appender.LOGFILE.layout.ConversionPattern=%-4r [%t] %-5p %c
- %m%n
```

Se debe indicar a la clase que inicializa el *logger* que la configuración se realiza mediante el *properties* y que se usa la clase *PropertyConfigurator* del paquete Log4j para configurarlo. La sentencia es la siguiente:

```
PropertyConfigurator.configure("log4j.properties");
```

Otra forma de configurar el *logger* es mediante un fichero XML. Es equivalente a la anterior y se utiliza `DOMConfigurator` para configurarlo. Un ejemplo de fichero XML de configuración es el siguiente:

```
     <?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE log4j:configuration SYSTEM "log4j.dtd">
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

  <appender name="Fichero" class="org.apache.log4j.FileAppender">
      <param name="File" value="A1.log" />
          <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern"
                  value="%t %-5p %c{2} - %m%n"/>
          </layout>
        </appender>

        <appender name="Consola" class="org.apache.log4j.ConsoleAppender">
          <layout class="org.apache.log4j.PatternLayout">
            <param name="ConversionPattern"
                  value="%d %-5p [%t] %C{2} (%F:%L) - %m%n"/>
          </layout>
        </appender>

        <root>
           <priority value ="debug"/>
           <appender-ref ref="Consola"/>
           <appender-ref ref="Fichero "/>
        </root>
</log4j:configuration>
```

La forma de inicializar esta configuración es:

```
      DOMConfigurator.configure("log4j.xml");
```

Normalmente, la elección de una forma u otra de configuración es transparente para el usuario. En algunos casos particulares, como en la configuración del *AsyncAppender*, sólo se puede utilizar el fichero XML.

6. Arquitectura

6.1. Introducción

Este capítulo describe la arquitectura del sistema propuesto para este estudio. Dicho sistema consta de dos aplicaciones, la principal es una aplicación Web de venta de libros por Internet. Los usuarios acceden a esta librería, consultan el catálogo, compran libros y sus transacciones se almacenan en una base de datos. La otra aplicación está destinada a los proveedores de la librería, los que, gracias al uso de servicios Web, pueden actualizar el *stock* almacenado en la base de datos.

6.2. Arquitectura de la aplicación Web

La aplicación Web está alojada en un servidor de páginas Web, Apache Tomcat, que se encarga de servir la aplicación al usuario e interactúa con la base de datos MySQL mediante el conector JDBC[4] de Java. La base de datos puede encontrarse en el mismo servidor que el servidor Tomcat o en uno diferente. Este aspecto es transparente al usuario.

A continuación se muestra cómo es la arquitectura en el desarrollo realizado, con un único servidor de aplicaciones y base de datos. Como se ha dicho, la base de datos puede alojarse en la misma máquina que el servidor de aplicaciones o en una diferente. En la figura siguiente está en una máquina diferente para poder diferenciarlos, pero realmente ambos sistemas se encuentran en el mismo servidor.

[4] JDBC (*Java Database Connectivity*) es una interfaz de programación que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java.

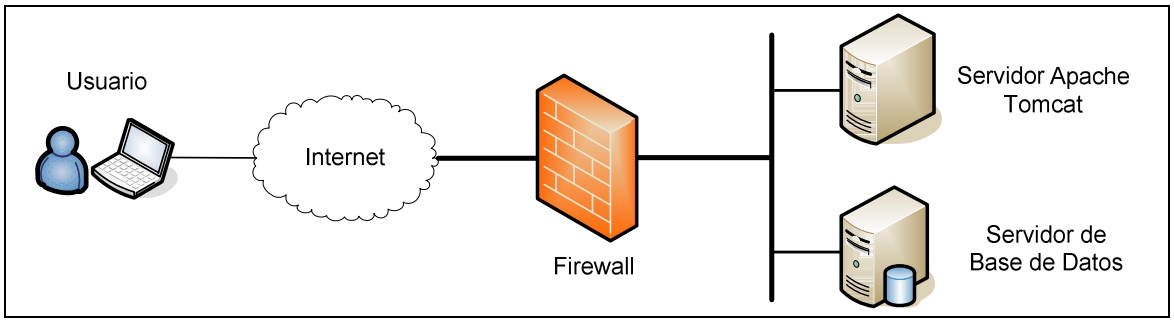


Figura 6-1. Arquitectura del sistema en un entorno de pruebas.

En un entorno empresarial, en el que el número de transacciones es muy elevado y hay continuos accesos a la base de datos, sería necesario reforzar esta arquitectura. Para desplegar esta tienda en un entorno en el que va a haber una fuerte carga transaccional y un elevado número de conexiones concurrentes habría que realizar un estudio de la carga esperada para poder dimensionar la cantidad de servidores de aplicaciones y de base de datos necesarios para hacer frente a dicha carga.

A continuación se muestra una aproximación de cómo sería la arquitectura en dicho entorno:

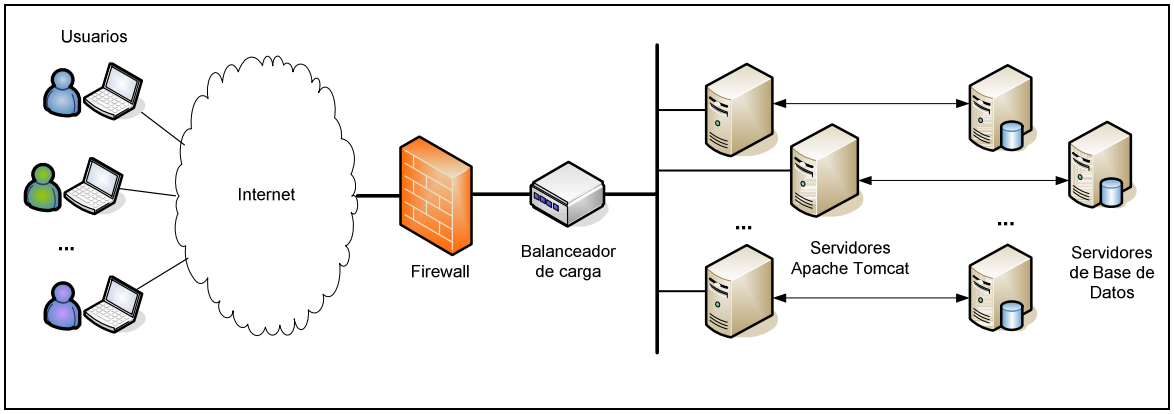


Figura 6-2. Arquitectura del sistema en un entorno de producción.

6.3. Arquitectura de la aplicación SOAP

La aplicación SOAP está alojada en un servidor de servicios Web Axis2. Este servidor puede estar instalado en la misma máquina donde está instalado el contenedor de *servlets* Apache Tomcat o en una diferente. En el caso de este proyecto, el servidor Axis2 sí está instalado en la misma máquina, aunque en un entorno comercial sería interesante separar ambos servidores

para que las transacciones de las editoriales las reciba un servidor dedicado y no interfiera esta carga en las operaciones de los clientes.

La siguiente figura ofrece una aproximación de cómo quedaría la arquitectura del sistema completo:

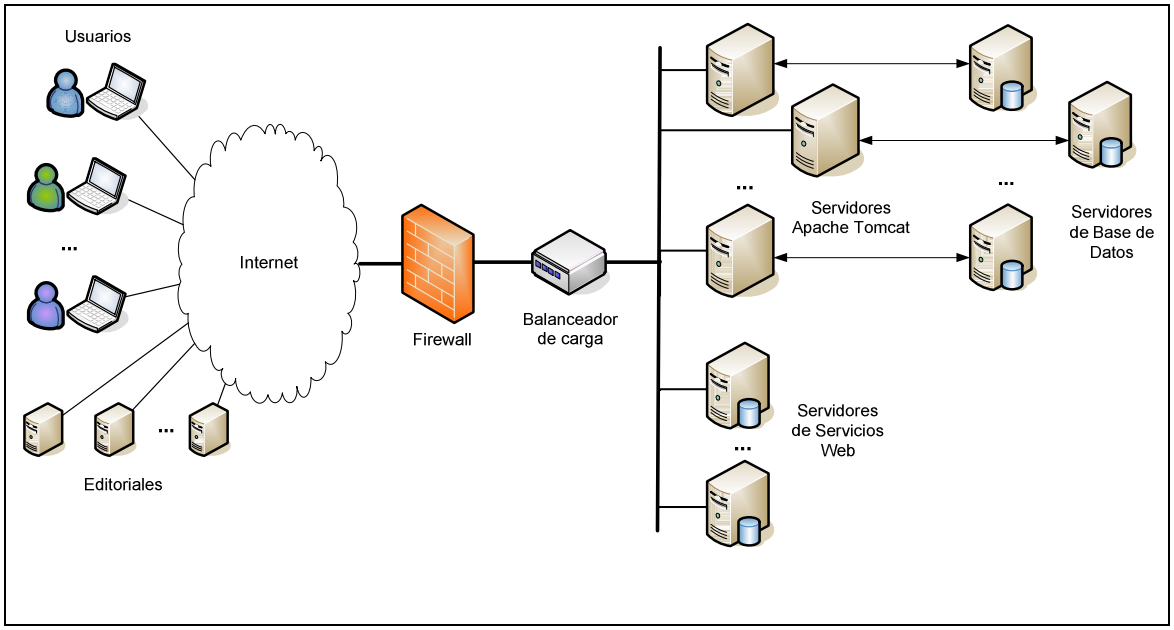


Figura 6-3. Arquitectura del sistema completo.

6.4. Requisitos funcionales del sistema

A continuación se muestran algunos de los requisitos funcionales que definen el sistema y que se corresponden en general con cualquiera de las librerías *on-line* que se pueden encontrar en el mercado. Algunos de los requisitos funcionales que definen la aplicación son los siguientes:

Registro de usuarios

La aplicación Web dispone de una página de registro de usuarios donde un usuario del sistema podrá proporcionar sus datos y registrarse en la aplicación. Esto le servirá para validarse en el sistema y poder realizar compras, así como para gestionar su archivo de favoritos.

Acceso de usuarios

La aplicación Web tiene una página de *login* en la cual un usuario dado de alta puede introducir sus credenciales y acceder como usuario registrado, con los privilegios correspondientes a un usuario registrado.

Favoritos

Los usuarios tienen la posibilidad de marcar los libros que deseen como favoritos, de tal forma que estén disponibles en futuras sesiones.

La aplicación Web dispone de una sección de favoritos que permite a los usuarios ver los libros que están registrados como favoritos. Esta opción sólo está disponible si el usuario se ha validado en el sistema. En el caso de que el usuario no esté validado, el intento de acceso a la sección de favoritos hará que se le envíe a la página de *login* para que se valide.

Resumen de compra

La aplicación Web dispone de una pantalla en la que se muestra a los usuarios el resumen de sus compras y el importe total de la compra en curso. Esta opción sólo está disponible si el usuario se ha validado en el sistema. En caso de que el usuario no esté validado, el intento de acceso al resumen de compra hará que se le envíe a la página de *login* para que se valide. Una vez se haya autenticado, se le mostrará dicha pantalla con el resumen de su compra.

Fichero de datos de *stock* de las editoriales

La aplicación editorial recibe un fichero XML que interpreta y, a partir de su contenido, crea el cliente Web necesario para invocar el servicio que actualiza el *stock* de la tienda en función de los valores que recibe.

6.5. Requisitos no funcionales del sistema

Los requisitos no funcionales son aquellos que no forman parte de la funcionalidad principal de la aplicación y que hacen referencia al entorno de desarrollo, entorno de ejecución, prestaciones, restricciones, etc. Este tipo de requisitos sí será más específico de la tienda que se describe y no tan común dentro de las tiendas *on-line* del mercado.

6.5.1. Entorno

A continuación se detalla el entorno sobre el cual están diseñadas e implementadas tanto la tienda electrónica como la editorial:

- La aplicación correrá en un servidor Tomcat 6.0.
- La base de datos que usará la aplicación es MySQL 5.0.
- Como aplicación de servicios Web se emplea Apache Axis2.
- El lenguaje de marcado a utilizar por las editoriales para enviar las actualizaciones de *stock* al almacen es XML.

6.5.2. Requisitos de usabilidad

El nivel de usabilidad de una aplicación informática indica el grado en el que el diseño de la misma facilita su manejo al usuario objetivo. En los requisitos de usabilidad de esta aplicación se intenta ver qué pasos necesita realizar el usuario para manejarla y la forma de facilitar su uso.

Los dos requisitos que se han tenido en cuenta en el diseño de la aplicación para conseguir una buena usabilidad son los siguientes:

- La aplicación dispone de unos menús sencillos e intuitivos que orientan al usuario a lo largo del proceso de navegación y compra.
- En la sección de compras se muestra al usuario un desglose de los productos que ha seleccionado para comprar y se le da la opción de borrar del resumen de compra los que no desee adquirir.

6.5.3. Requisitos de apariencia

En todas las aplicaciones comerciales, como esta, un requisito muy importante es que el usuario disponga de un interfaz de uso agradable. A continuación se indican algunos requisitos de apariencia que se han tenido en cuenta en el diseño de la aplicación:

Se ha buscado que la aplicación tenga un diseño sencillo para evitar que se llegue a incomodar al usuario con colores estridentes o recargados.

Al ser una aplicación destinada a un grupo muy heterogéneo de usuarios, debe ser capaz de ser visible al menos en los navegadores más importantes. El navegador más utilizado es Internet Explorer, de Microsoft, pero cada vez está perdiendo más usuarios en beneficio de otros

navegadores, como Mozilla Firefox, Opera o, para los usuarios de Macintosh, Safari. Resulta muy desalentador ver aplicaciones comerciales que sólo están disponibles para los usuarios de Internet Explorer, lo que en algunos casos implica la pérdida de clientes. Para ello, en el diseño de la tienda sólo se han utilizado *tags* HTML con atributos estándar y hojas de estilo CSS. Además, con las hojas de estilo se logra un diseño más fácilmente mantenible y se tiene posibilidad de cambiar el aspecto completo de la aplicación solamente cambiando la hoja de estilo.

6.5.4. Requisitos de rendimiento

Al ser una aplicación comercial, que en un entorno de producción recibirá un gran numero de transacciones simultáneas, se deberá garantizar alta disponibilidad y rendimiento suficiente. En el caso actual solamente se han tenido en cuenta requisitos referentes a la conexión con la base de datos, pero en un entorno empresarial real habría que tener varias máquinas sirviendo la aplicación a la vez, así como un balanceador de carga para distribuir el trabajo entre las mismas.

Los dos requisitos de rendimiento que se han tenido en cuenta son los siguientes:

- Las conexiones a la base de datos de la aplicación Web están gestionadas mediante un *pool* de conexiones.
- Las conexiones a la base de datos de la aplicación SOAP están controladas por el propio servicio.

6.5.5. Requisitos de seguridad

El almacenamiento de las contraseñas de los usuarios en la base de datos se realiza de forma segura. Al darse de alta el usuario, éste elige un identificador y una contraseña que servirán para validarse en el sistema. Las contraseñas se almacenan en la base de datos encriptadas mediante en algoritmo de cifrado MD5. Este algoritmo consiste en una función de cifrado que recibe una cadena de texto como entrada y devuelve un número de 128 bits.

La principal ventaja de este tipo de algoritmos es la imposibilidad computacional de reconstruir la cadena original a partir del resultado, así como la imposibilidad de encontrar dos cadenas de texto diferentes que generen el mismo resultado. Como desventaja destaca la posibilidad de averiguar la cadena original mediante un ataque de fuerza bruta.

Cuando el usuario introduce sus credenciales en el sistema, se cifra la contraseña introducida y se compara con la almacenada en la base de datos. Si éstas coinciden se autentica al usuario en el sistema.

De esta forma se consigue que las contraseñas no se almacenen en claro, pero para completar la seguridad se recomienda el envío de las mismas a través de la red mediante un canal seguro. No se ha definido en el alcance del proyecto el uso de SSL[5] para realizar las transacciones, pero si esta aplicación se utilizara en un entorno real este paso sería fundamental.

[5] SSL (Secure Sockets Layer) es un protocolo que proporciona conexiones seguras por una red insegura, comúnmente Internet.

7. Implementación y funcionamiento

7.1. Introducción

El proyecto consiste en una tienda electrónica utilizando las bibliotecas Struts y SOAP, de las que se ha hablado con detalle anteriormente. A continuación se explica cómo se ha realizado dicha tienda y qué decisiones de diseño se han tomado. Asimismo, se verá de una forma global cómo interactúan cada una de las tecnologías empleadas con Struts y SOAP formando un conjunto.

7.2. Descripción del proyecto

La tienda consta de varias secciones claramente diferenciadas.

- **Búsquedas**: existe un apartado para las búsquedas con el fin de facilitar la búsqueda de libros a los usuarios.
- **Favoritos**: los usuarios registrados tienen la opción de seleccionar y marcar ciertos libros como favoritos, de forma que éstos estén disponibles en sucesivas conexiones y no sea necesario buscarlos de nuevo.
- **Carrito de la compra**: esta es la funcionalidad normal de compra en una tienda *on-line* y se conoce como carrito de la compra. Una vez que el usuario está preparado para efectuar la compra, se le pide la validación de sus credenciales en el sistema y se le orienta en el proceso de compra. El carrito de la compra sólo tiene ámbito de sesión, esto es, durante el tiempo que el usuario está conectado.

Además de estas funcionalidades para clientes, la tienda también dispone de un servicio especial para las editoriales. Dicha funcionalidad permite a las editoriales actualizar su *stock* en la tienda de una forma automática a medida que se actualiza su propio *stock* en los almacenes. Las editoriales emplean una pequeña utilidad que se conecta con la tienda mediante el protocolo SOAP y actualiza directamente la base de datos. De esta forma se garantiza una gestión rápida y automática del *stock*, sin intermediarios. Además, al ser una pequeña aplicación que se conecta

mediante mensajes SOAP, no es necesario disponer de un ordenador con muchos recursos para enviar dichas actualizaciones, puesto que, como ya se ha expuesto, SOAP es un protocolo muy ligero.

7.3. Usuarios de acceso

En la aplicación se distinguen tres tipos de usuario de acceso con distintas funcionalidades y privilegios.

Invitado. Tiene acceso a todos los contenidos de la aplicación, pero no puede realizar ninguna compra. En caso de querer hacerla, se le ofrece la posibilidad de registrarse.

Autenticado. Similar al invitado pero con la posibilidad de comprar y de guardar un historial con sus libros favoritos.

Editorial. Puede enviar a la aplicación documentos XML que contendrán los libros que van a ofertar e información sobre ellos. Mediante una llamada RPC al sistema se actualizarán las entradas de la base de datos con los libros ofertados por la editorial y se pondrán a la venta de inmediato.

Se han creado estos usuarios porque se considera que son suficientes para mostrar la funcionalidad de la aplicación, pero para una aplicación empresarial se podría ampliar este número introduciendo la figura de un administrador, que gestionaría la aplicación, un gestor de almacen, que verificaría los pedidos y se encargaría de realizar los envíos, etc.

7.4. El modelo de datos

A continuación se muesta el modelo de datos empleado para la realización de la tienda electrónica:

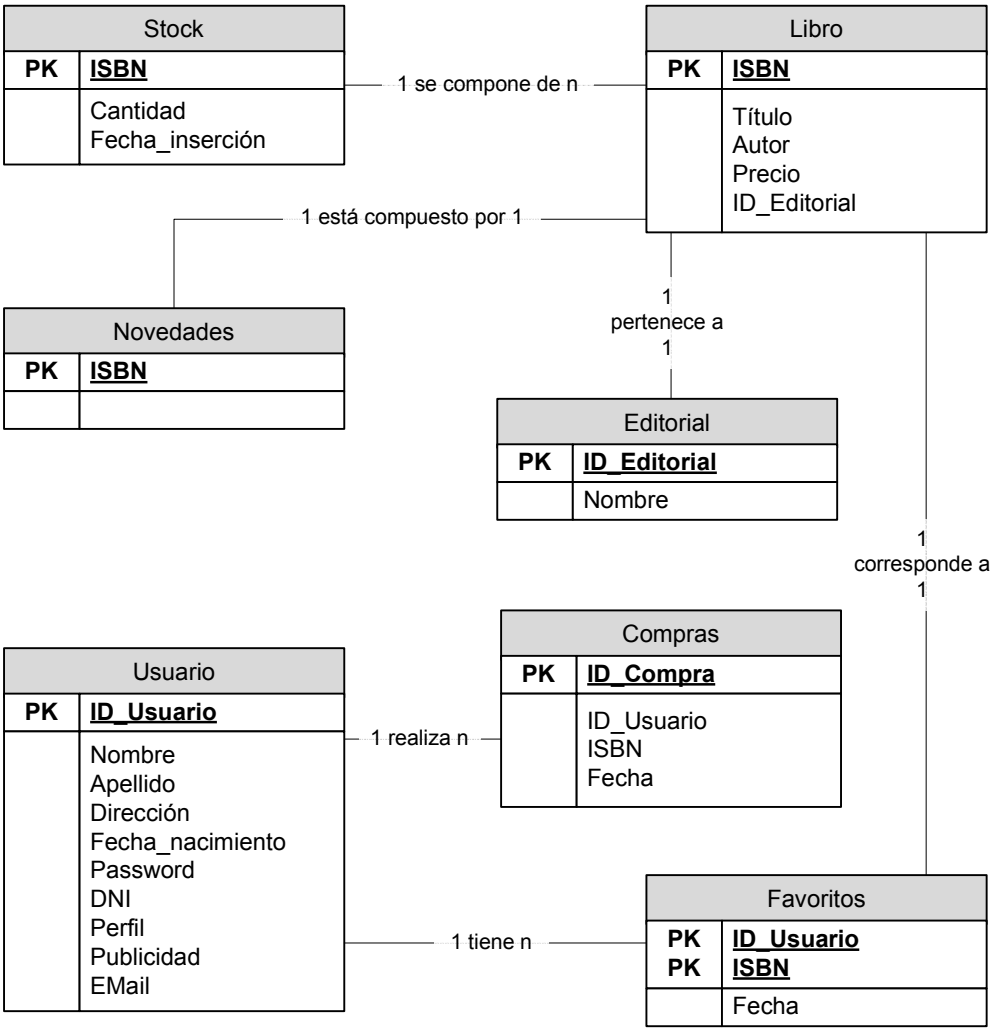


Figura 7-1. Modelo de datos.

7.5. Decisiones de diseño en la aplicación Web

7.5.1. El controlador en la aplicación Web

Como se ha visto en el capítulo 2, en una aplicación Struts se encuentran muy bien definidos el modelo, la vista y el controlador. El controlador se define por el *framework* mediante el fichero *struts-config.xml* y las clases *Action*, pero depende del desarrollador hacer una buena implementación de los mismos. A continuación se va a explicar cómo se ha implementado el controlador en la aplicación Web.

El componente más importante de los que gestionan el controlador es la clase *Action*. Esta clase actúa como puente entre una acción de usuario del lado del cliente y una operación de negocio. Si se emplea la clase *Action* tal como la provee Struts, será necesario crear una de estas

clases por cada opción de navegación de la aplicación y cada una de ellas tendrá su método `execute`, que se invocará al hacer el *submit* en la página jsp. Para una aplicación transaccional, en la que el número de posibilidades de navegación es relativamente elevado, disponer de una clase por cada opción de navegación dificulta mucho el mantenimiento de la aplicación.

Con el fín de evitar esto, en la aplicación Web diseñada, se ha implementado una clase *MainAction* de la que heredan todas las demás *Action*. El código de la clase *MainAction* es muy sencillo, hereda de la clase *DispatchAction* y es la única que contiene el método `execute`. Cuando se invoca dicho método, se llama al `execute` de la clase padre pasándole como parámetros el *ActionMapping*, el *ActionForm* asociado, la `request` y la `response`. El *ActionForm* asociado es un formulario genérico del que heredan el resto de formularios de la aplicación. La principal característica de este formulario es que contiene un atributo *accion* que almacena la acción asociada a cada evento del formulario.

El método execute de la clase *MainAction* queda de la siguiente manera:

```
public  ActionForward execute(ActionMapping mapping,
                              ActionForm form,
                              HttpServletRequest request,
                              HttpServletResponse response) throws
Exception{
      try {
          httpRequest=request;
          httpSesion=request.getSession();
          ActionForward forward = mapping.findForward(FWD_ERROR);
          try {
                   GenericForm gForm = (GenericForm) form;
                   forward = super.execute(mapping,form,request,response);
                   return (forward);
          }
          catch (Exception e){
              e.printStackTrace();
              return (mapping.findForward(FWD_ERROR));
          }
      }catch (Exception e) {
          e.printStackTrace();
          return (mapping.findForward(FWD_ERROR));
      }
    }
```

Si no existiera la clase *MainAction*, cada uno de los *action* de la aplicación tendría que heredar directamente de *DispatchAction* y el único método que tendría sería el método `execute`, con lo que, como se ha dicho, habría que tener una clase por cada evento.

La forma en que está organizada la aplicación es la siguiente: existe una clase *Action* por cada uno de los apartados de la aplicación. Hay un *Action* y un *Form* que gestionan la parte de compras, otros que gestionan la parte de *login*, otro par para la parte de novedades, etc.

A continuación se muestra cómo son las clases que gestionan la parte de compras.

Para la parte de compras se tiene una clase de tipo *Action*, llamada `ActionCompras`, y otra de tipo *Form*, que se llama `FormCompras`. La clase `ActionCompras` hereda de *MainAction* y contiene un método por cada una de las operaciones que se pueden dar en la parte de compras. En cada uno de estos métodos se invocan las operaciones de negocio correspondientes y se devuelve el *ActionForward*, que controla la siguiente transición a mostrar. Este es el esqueleto de la clase *ActionCompras*:

```
public class ActionCompras extends MainAction{

public Logger  logger =
      Logger.getLogger("es.pfc.shop.actions.ActionCompras.class");

  public ActionForward event_mostrar_compras (ActionMapping mapping,
            ActionForm form,
            HttpServletRequest request,
            HttpServletResponse response) throws Exception;


  public ActionForward event_remove_item (ActionMapping mapping,
            ActionForm form,
            HttpServletRequest request,
            HttpServletResponse response) throws Exception;


  public ActionForward event_comprar_item (ActionMapping mapping,
            ActionForm form,
            HttpServletRequest request,
            HttpServletResponse response) throws Exception;


  public ActionForward event_validar_compra (ActionMapping mapping,
            ActionForm form,
            HttpServletRequest request,
            HttpServletResponse response) throws Exception;
}
```

Como se puede observar, existen varios métodos cuya denominación indica el tipo de operaciones que se van a realizar dentro del mismo. Por ejemplo, el método `event_remove_item` borra un elemento del carrito de la compra. Cuando se invoca desde la página jsp, el formulario asociado a la clase *ActionCompras*, es decir, la clase *FormCompras*,

debe pasarle, además de todos los atributos necesarios para la lógica de negocio, el atributo *accion* con el valor del método que se va a invocar. En este caso:

```
String accion = "event_remove_item"
```

Asociándole como parámetro la acción asociada, la clase *ActionFormat* sabe que el método que se tiene que ejecutar es el que viene en la acción y no otro. De esta forma podemos agrupar varios métodos relativos a una misma operativa de la tienda en una sola clase *Action* en lugar de tener varias clases por cada operación de negocio.

Se ha tomado esta decisión de diseño porque de esta forma se crean menos clases y éstas agrupan funcionalidades, con lo que el mantenimiento de la aplicación es más sencillo.

7.5.2. El acceso a la base de datos

A continuación se verá cómo se han modelado los accesos a la base de datos y cómo se han gestionado las conexiones.

Las conexiones a la base de datos están gestionadas por un *pool* de conexiones. Se denomida *pool* de conexiones al grupo de de conexiones abiertas a una base de datos que se emplean para acceder a la misma, de manera que pueden ser reutilizadas para realizar múltiples consultas o actualizaciones.

Crear conexiones a la base de datos es una operación muy costosa, además de que existe el riesgo de que el desarrollador no cierre la conexión y se mantengan conexiones abiertas innecesariamente, con la consiguiente penalización en el rendimiento. Para evitar esto, la aplicación dispone de un *pool* de conexiones a la base de datos. Cuando se inicializa el *pool* se crea un número determinado de conexiones con la base de datos que se mantienen en memoria a la espera de ser utilizadas. Si la aplicación necesita una conexión, se la solicita al gestor del *pool*, que le da una de las que tiene disponibles. La aplicación usa esta conexión y al finalizar, en lugar de destruirla, se la devuelve al gestor del *pool*, que la mantiene en memoria a la espera de ser reutilizada de nuevo. Con esto, la aplicación hace un esfuerzo inicial en la creación de conexiones pero durante su funcionamiento normal, cuando más interesa no penalizar el rendimiento, se apreciará una gran mejora.

En la tienda *on-line* el *pool* de conexiones está gestionado por el contenedor de *servlets*, Apache Tomcat. Además de las ventajas que tiene crear este tipo de gestores, en la aplicación se ha elegido el que provee Tomcat porque realiza un control de conexiones abandonadas. En una

aplicación Web, el hecho de que haya *ResultSet* o *Statements* que no se cierren nunca pueden llegar a provocar que haya conexiones que se queden en un estado en el que no se pueden reutilizar. Esto puede ocasionar que la aplicación necesite una nueva conexión y el proceso falle por no encontrarse conexiones disponibles. El gestor de conexiones de Apache Tomcat soluciona este problema ya que puede ser configurado para que detecte y recupere las conexiones abandonadas. Además, se puede configurar el tiempo que una conexión puede estar inactiva sin que se considere que está abandonada.

Como se ha visto, este *pool* de conexiones es muy funcional y además es muy sencillo de crear y configurar. A continuación se muestra la directiva incluida en el fichero de configuración de Tomcat, necesaria para activar el gestor de conexiones y configurarlo:

```
<Resource name="jdbc/ElephantDB"
          auth="Container" type="javax.sql.DataSource"
          maxActive="100"
          maxIdle="30"
          maxWait="10000"
          removeAbandoned = "true"
          username="pfc"
          password="pfc"
          driverClassName="com.mysql.jdbc.Driver"
          url="jdbc:mysql://localhost:3306/pfc?autoReconnect=true"/>
```

De esta forma el *pool* de conexiones está disponible para cualquier aplicación Web desplegada en el servidor Tomcat. Se pueden definir tantos gestores como se deseen y para diferentes bases de datos. En este caso se ha creado para MySQL, pero también se puede crear para bases de datos Oracle o PostgreSQL, por ejemplo.

Para que una aplicación Web concreta utilice un *pool* de conexiones, hay que indicarlo en el fichero *web.xml* de la aplicación:

```
<description>MySQL DataSource</description>
  <resource-ref>
    <description>DB Connection</description>
    <res-ref-name>jdbc/ElephantDB</res-ref-name>
    <res-type>javax.sql.DataSource</res-type>
    <res-auth>Container</res-auth>
  </resource-ref>
```

Con esto, el recurso está diponible para ser utilizado por la aplicación. Al iniciarse la aplicación se obtiene la fuente de datos correspondiente y cuando ésta necesita una conexión, la solicita al gestor, que le pasa el recurso inmediatamente. La forma de pedir una conexión se indica en el siguiente código de ejemplo:

```
Context ctx = new InitialContext();
DataSource fuenteDatos =
  (DataSource) ctx.lookup("java:comp/env/jdbc/ElephantDB");
Connection conn = fuenteDatos.getConexion();
```

Como ventaja añadida, hay que considerar que con esta forma de gestionar las conexiones se consigue que el desarrollador se abstraiga por completo de la base de datos y sea el administrador del servidor de aplicaciones el que conozca las claves de acceso y se encargue de la gestión de la misma.

7.5.3. El patrón de acceso a los datos

Las operaciones en la base de datos se modelan mediante objetos DAO, *Database Access Objects*. El patrón DAO encapsula la forma de acceder a la fuente de datos y las operaciones sobre dicha fuente. De esta forma el programador no necesita conocer los detalles del acceso a los datos o de la base de datos a la que accede y sólo debe preocuparse de los datos que necesita.

Como muestra la figura siguiente, se dispone de una interfaz que define los métodos que va a tener el objeto DAO. Habitualmente se definen métodos comunes para la gestión de datos, como listar, borrar, actualizar, etc. Estos métodos suelen identificar las acciones a ejecutar sobre los datos. Los pasos que se realizan son los siguientes:

1. El objeto DAO es instanciado por el cliente.
2. El cliente solicita los datos al objeto DAO.
3. El DAO a su vez solicita los datos a la fuente.
4. Por cada fila recibida, el DAO crea un objeto de transferencia, `TransferObject`, que será un *bean* que tiene como atributos los de la *query* solicitada.
5. Se devuelven los objetos de transferencia al cliente.

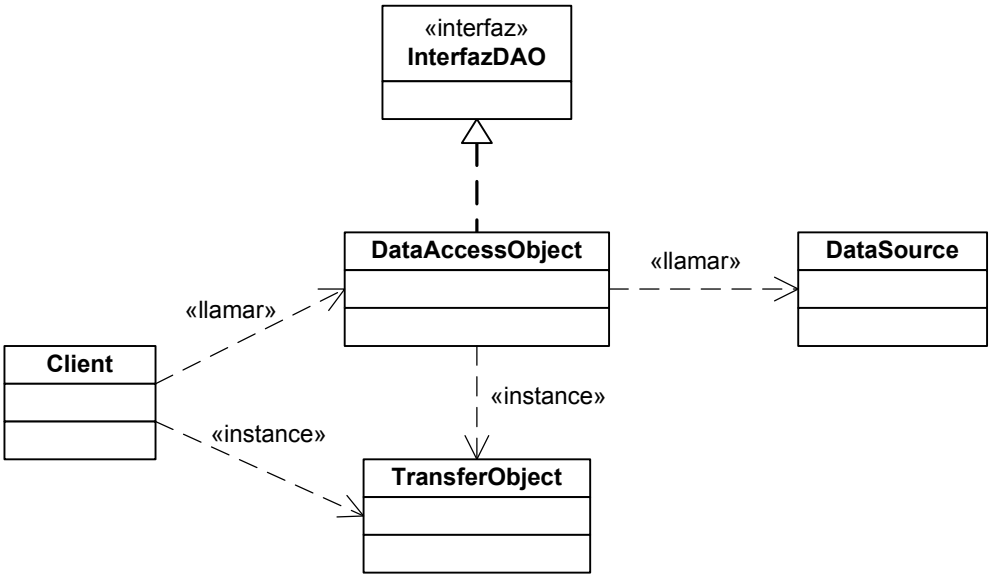


Figura 7-2. Modelo de procesamiento DAO.

Una de las ventajas de utilizar este patrón para modelar el acceso a los datos es que habilita la transparencia en los accesos. Los objetos de negocio pueden acceder a las fuentes de datos sin conocer detalles específicos de las mismas. Este acceso es transparente porque los detalles de la implementación están encapsulados dentro del objeto DAO.

Además, se facilita el mantenimiento de la aplicación y futuras migraciones de la base de datos. Con el empleo de este patrón el código generado es muy sencillo e intuitivo para el desarrollador, lo que facilita el mantenimiento de la aplicación y el desarrollo de nuevas versiones.

Por otro lado, el empleo del modelo DAO también tiene desventajas, como que requiere un mayor tiempo tanto en el diseño como la implementación por la existencia de una capa extra entre la lógica de negocio y los datos que ésta emplea.

7.5.4. Tiles

Struts Tiles es una funcionalidad que provee Struts para crear plantillas y variar el contenido de las diferentes secciones que componen una página. En una aplicación de comercio electrónico es muy común que haya partes que permanezcan estáticas, como el logo de la marca comercial, el pie o los menús, y otras que vayan variando de unas páginas a otras, como el cuerpo de la página.

A continuación se muestra un esquema de cómo está organizada una página Web cualquiera de la aplicación:

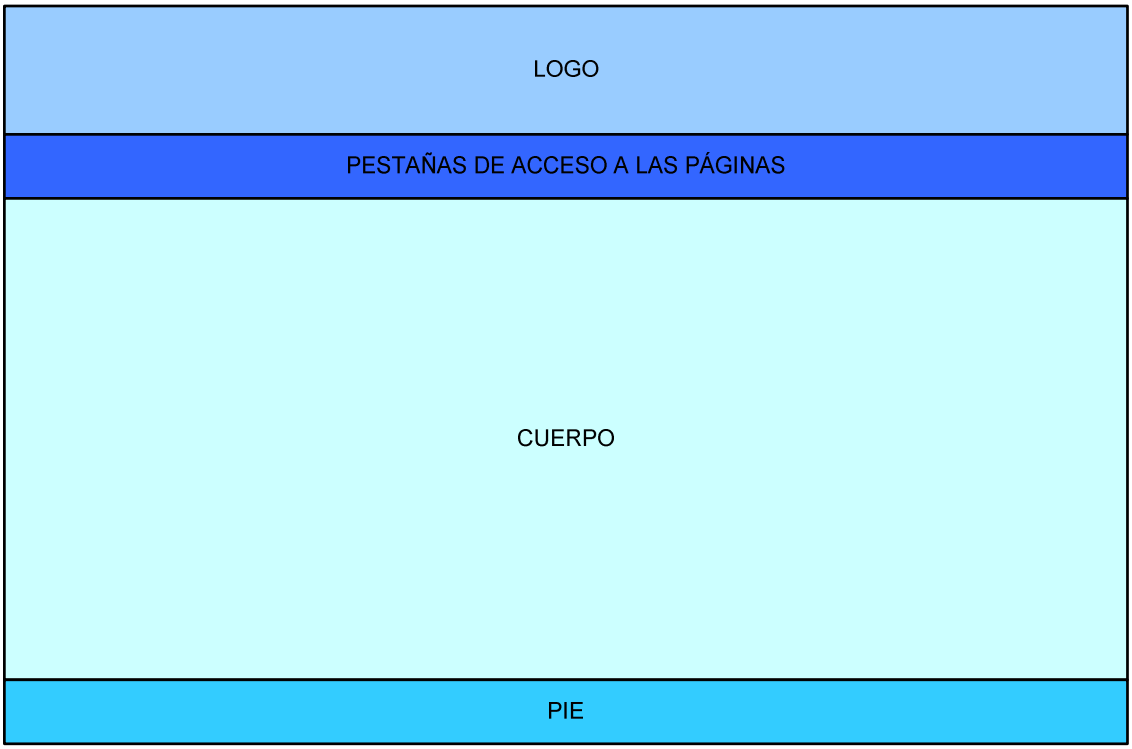


Figura 7-3. Partes en las que se divide una página de la aplicación.

Para reproducir el diseño de la página se ha creado un archivo jsp que se utiliza como plantilla y que contiene todos los elementos mostrados en la figura anterior. El código de la plantilla se muestra a continuación:

```
<%@ taglib uri="/tags/struts-bean" prefix="bean" %>
<%@ taglib uri="/tags/struts-html" prefix="html" %>
<%@ taglib uri="/tags/struts-logic" prefix="logic" %>
<%@ taglib uri="/tags/struts-tiles" prefix="tiles" %>
<html>
<head>
<link href="css/tabs.css" rel="stylesheet" type="text/css">
<link href="css/tablas.css" rel="stylesheet" type="text/css">
<link href="css/style.css" rel="stylesheet" type="text/css">
<script language="Javascript" src="script/acciones.js"></script>
<title><tiles:getAsString name="titulo"/> </title>
</head>
<body>
  <table border="0" width="100%">
      <tr>
          <td>
              <tiles:insert attribute="logo" ignore="true"></tiles:insert>
```

```
            </td>
        </tr>
        <tr>
            <td>
                <tiles:insert attribute="tabs" ignore="true"></tiles:insert>
            </td>
        </tr>
        <tr>
            <td>
                <tiles:insert attribute="body" ignore="true"></tiles:insert>
            </td>
        </tr>
        <tr>
            <td>
                <tiles:insert attribute="pie" ignore="true"></tiles:insert>
            </td>
        </tr>
    </table>
</body>
</html>
```

Esta página contiene los cuatro elementos que se han visto en la figura anrterior e importa las hojas de estilo y el fichero con contenido *javascript*. Al ser la página principal la que contiene las hojas de estilo y las funciones *javascript* éstas están disponibles para el resto de páginas. Aquí se puede observar una ventaja de utilizar Tiles, ya que se evita el tener que importar estos elementos en cada una de las páginas.

Tanto el logo de la aplicación como el pie y los menús en pestañas permanecen estáticos durante toda la aplicación, por lo que solamente hay que darlos de alta una vez incluyéndolos en la plantilla. A continuación se explican los pasos que se han llevado a cabo.

En primer lugar hay que indicar que la aplicación va a utilizar Tiles, para ello se incluye la siguiente directiva en el fichero *struts-config.xml*:

```
<plug-in className="org.apache.struts.tiles.TilesPlugin" >
  <set-property property="definitions-config"
        value="/WEB-INF/tiles-defs.xml" />
  <set-property property="moduleAware" value="true" />
</plug-in>
```

Aquí se indica el fichero en el que se van a definir tanto las plantillas como las páginas. En el fichero *tiles-defs.xml* se ha definido la plantilla correspondiente a la página mostrada anteriormente y se han incluido las partes estáticas.

```
<tiles-definitions>

    <definition name="base.definition" path="/pages/plantilla.jsp">
      <put name="logo" value="/pages/logo.jsp"></put>
```

```
      <put name="tabs" value="/pages/tabs.jsp"></put>
      <put name="pie" value="/pages/pie.html"></put>
    </definition>
  .
  .
  .
</tiles-definitions>
```

Como se puede ver, se insertan las partes estáticas pero se dejan libres las correspondientes al cuerpo de la página y al título de la misma, que irán variando durante toda la aplicación.

El resto de páginas que se van a mostrar hay que definirlas una a una e irán heredando de `base.definition`, de este modo contendrán su propio *cuerpo* y *título* y contendrán todos los elementos comunes. A continuación se muestra un ejemplo de la definición de algunas de las páginas empleadas en la aplicación:

```
<tiles-definitions>
  .
  .
  .
    <definition name="page.novedades" extends="base.definition">
      <put name="titulo" value="Elephant Books - Novedades"></put>
      <put name="body" value="/pages/novedades.jsp"></put>
    </definition>
    <definition name="page.favoritos" extends="base.definition">
      <put name="titulo" value="Elephant Books - Favoritos"></put>
      <put name="body" value="/pages/favoritos.jsp"></put>
    </definition>
  .
  .
  .
</tiles-definitions>
```

Como se puede ver, heredan los elementos comunes e introducen sus propios elementos, con lo que queda completa la página a mostrar.

El resultado de la página es el siguiente:

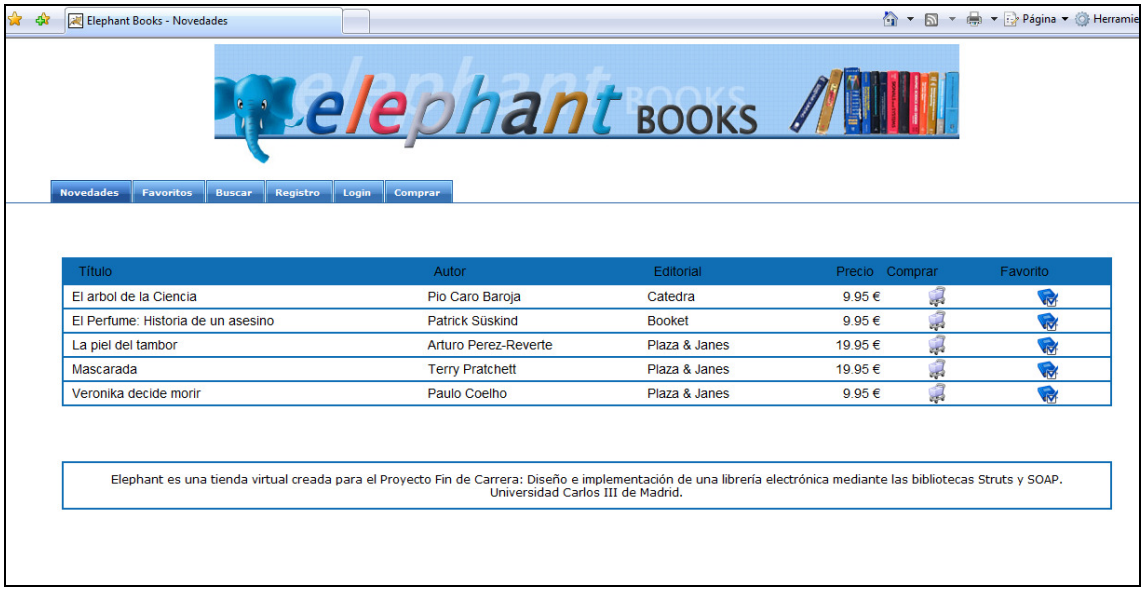


Figura 7-4. Página de la aplicación Web.

7.6. Decisiones de diseño en la aplicación SOAP

7.6.1. El servidor de servicios Web: Axis2

Axis2 es el motor de de servicios Web de Apache. Se puede descargar desde su página Web, donde se encuentran los siguientes tipos de distribución:

- *Standard Binary Distribution*: Es la versión completa de Axis2. Puede ejecutarse, a diferencia de Axis, como un servidor independiente o generar una aplicación Web para desplegar en un contenedor de *servlets*. Contiene, además, las herramientas *wsdl2java*, *java2wsdl* y los ejemplos de ayuda.
- *Source Distribution*: Contiene el código fuente, lo que permite generar una versión ejecutable con Maven[6].
- *WAR (Web Archive) Distribution*: Es una versión preparada para ser desplegada en un contenedor de *servlets* como una aplicación Web.
- *Documents Distribution*: Se trata de un fichero zip con toda la documentación.

Para este proyecto se ha utilizado la primera opción, *Standard Binary Distribution*, que se debe descomprimir en un directorio de trabajo. Para que todo funcione correctamente se debe

[6] Maven es una herramienta de *software* para la gestión y compresión de proyectos Java. Basándose en un fichero XML permite compilar, ejecutar *tests* unitarios o generar distribuciones. Pertenece a la Apache Software Foundation.

crear la variable de entorno del sistema `AXIS2_HOME` para que apunte a la ruta donde se ha descomprimido.

La estructura de carpetas que se observa es la siguiente:

- *bin*: contiene *scripts* para ejecutar Axis2 como un servidor *standalone*, *axis2server*, y las herramientas *wsdl2java* y *java2wsdl*.
- *conf*: incluye el fichero de configuración *axis2.xml*.
- *lib*: contiene las bibliotecas que utiliza Axis2.
- *repository*: contiene los servicios y módulos disponibles.
- *samples*: contiene códigos de ejemplo
- *webapp*: contiene la aplicación Web de administración de Axis2. Se incluye sólo para la generación de una distribución WAR de Axis2.

El servidor de servicios Web de Axis2 puede ejecutarse en dos modos, como aplicación Web desplegada en un contenedor de *servlets* o como servidor *standalone* gracias a un pequeño servidor que viene integrado con la distribución.

Para iniciar el servidor en modo *standalone* se puede usar el *script axis2server*, que está dentro de la carpeta *bin*. Una vez iniciado, despliega todos los servicios que contiene la carpeta *repository*. El servidor escucha por defecto en el puerto 8080, aunque se puede cambiar en el fichero *axis2.xml*, que está dentro de la carpeta *conf*, mediante la siguiente directiva:

```
<transportReceiver name="http"
        class="org.apache.axis2.transport.http.SimpleHTTPServer">
  <parameter name="port">8880</parameter>
</transportReceiver>
```

Accediendo a la siguiente dirección local *http://localhost:8880/axis2/services* se puede comprobar que el servidor ha arrancado correctamente y consultar qué servicios tiene desplegados.

Axis2 permite desplegar servicios Web sin tener que parar y arrancar el servidor, para ello basta con copiar el servicio Web empaquetado en formato aar (*Axis Archive*) a la carpeta *webapp*. Un fichero *.aar* contiene los ficheros *.class* correspondientes a las clases del servicio y el fichero *services.xml*.

7.6.2. Modelo WSDL utilizado

De todas las formas disponibles para crear un servicio Web se ha optado por utilizar el generador de código de Axis2. Axis2 dispone de dos utilidades para la generación de un servicio Web, una es Java2WSDL, que crea el fichero WSDL a partir de una implementación Java del servicio, y la otra es WSDL2Java que, de manera contraria, crea el esqueleto Java del servicio a partir de una definición WSDL.

Para la elaboración de este proyecto se ha elegido la segunda opción, en la que a partir de un fichero WSDL, que define todos los detalles de un servicio Web, como las operaciones, el formato y tipo de los mensajes o el protocolo de transporte utilizado, se genera el esqueleto del servicio en código Java. De esta forma se evita que la generación del fichero descriptor del servicio dependa de la implementación.

Un buen diseño del fichero WSDL es fundamental a la hora de crear un servicio Web, ya que de esto dependerá que no haya problemas de interoperabilidad aunque se empleen tecnologías diferentes a la del servicio para crear los clientes.

El primer paso para desarrollar un servicio Web es definir el formato de los mensajes XML de cada operación. Para ello se ha creado un *schema* XSD donde se definen los elementos de entrada y salida de las operaciones del servicio. A continuación se puede ver el *schema* asociado al servicio Web *Editorial*.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns:tns="http://www.example.org/editorialSchema/messages"
  xmlns="http://www.w3.org/2001/XMLSchema"
  targetNamespace="http://www.example.org/editorialSchema/messages">
  <element name="updateStockRequest">
      <complexType>
          <sequence>
              <element name="ISBN" type="string" minOccurs="0"/>
              <element name="Cantidad" type="string" minOccurs="0"/>
          </sequence>
      </complexType>
  </element>
  <element name="updateStockResponse">
      <complexType>
          <sequence>
              <element name="status" type="string" minOccurs="0"/>
          </sequence>
      </complexType>
  </element>
</schema>
```

Como se puede ver, se trata de un *schema* muy sencillo, se definen los mensajes de entrada y salida correspondientes a cada método que se va a invocar en el destino. El método `updateStock` se compone de una *request*, que es el mensaje de entrada que va a recibir el servicio y una *response*, que es el mensaje que va a devolver el servicio al cliente.

En este caso, la *request* lleva asociados dos parámetros, el ISBN del libro a actualizar y la cantidad nueva de *stock* disponible. La *response* solamente lleva un parámetro, que será el que reciba el cliente, en este caso el estado de la operación, de tal forma que el cliente sepa si la invocación al servicio se ha hecho correctamente.

Una vez que se han definido uno o varios *schemas* hay que crear el fichero WSDL que completa la definición del servicio Web, añadiendo una serie de parámetros correspondientes al servicio. A continuación se muestra el fichero WSDL del servicio *Editorial*:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<wsdl:definitions xmlns:soap="http://schemas.xmlsoap.org/wsdl/soap/"
  xmlns:tns="http://www.example.org/editorial/"
  xmlns:wsdl="http://schemas.xmlsoap.org/wsdl/"
  xmlns:msg="http://www.example.org/editorialSchema/messages"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema"
      name="editorial"
      targetNamespace="http://www.example.org/editorial/">
  <wsdl:types>
    <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
      <xsd:import
              namespace="http://www.example.org/editorialSchema/messages"
              schemaLocation="editorialSchema.xsd">
      </xsd:import>
    </xsd:schema>
  </wsdl:types>
  <wsdl:message name="updateStockRequest">
    <wsdl:part element="msg:updateStockRequest" name="parameters"/>
  </wsdl:message>
  <wsdl:message name="updateStockResponse">
    <wsdl:part element="msg:updateStockResponse" name="parameters"/>
  </wsdl:message>
  <wsdl:portType name="editorial">
    <wsdl:operation name="updateStock">
      <wsdl:input message="tns:updateStockRequest"/>
      <wsdl:output message="tns:updateStockResponse"/>
    </wsdl:operation>
  </wsdl:portType>
  <wsdl:binding name="editorialSOAP" type="tns:editorial">
    <soap:binding style="document"
      transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="updateStock">
      <soap:operation
soapAction="http://www.example.org/editorial/updateStock"/>
      <wsdl:input>
```

```
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
  </wsdl:binding>
  <wsdl:service name="editorial">
    <wsdl:port binding="tns:editorialSOAP" name="editorialSOAP">
      <soap:address location="http://www.example.org/"/>
    </wsdl:port>
  </wsdl:service>
</wsdl:definitions>
```

Los elementos más importantes de este archivo WSDL son los siguientes.

- **xsd:schema**: se utiliza para importar un *schema*. En este caso se importa el *schema editorialSchema.xsd*, del que se ha hablado anteriormente.

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
  <xsd:import namespace="http://www.example.org/editorialSchema/messages"
     schemaLocation="editorialSchema.xsd">
  </xsd:import>
</xsd:schema>
```

- **wsdl:message**: define los mensajes correspondientes a cada operación del servicio Web.

```
<wsdl:message name="updateStockRequest">
    <wsdl:part element="msg:updateStockRequest" name="parameters"/>
</wsdl:message>
```

- **wsdl:portType**: es el interfaz en el que se definen las operaciones del servicio Web.

```
<wsdl:portType name="editorial">
    <wsdl:operation name="updateStock">
      <wsdl:input message="tns:updateStockRequest"/>
      <wsdl:output message="tns:updateStockResponse"/>
    </wsdl:operation>
</wsdl:portType>
```

- **wsdl:binding**: especifica el protocolo, el tipo de servicio Web y los formatos de las operaciones de un *PortType* concreto.

```
<wsdl:binding name="editorialSOAP" type="tns:editorial">
    <soap:binding style="document"
          transport="http://schemas.xmlsoap.org/soap/http"/>
    <wsdl:operation name="updateStock">
      <soap:operation
            soapAction="http://www.example.org/editorial/updateStock"/>
      <wsdl:input>
```

```
        <soap:body use="literal"/>
      </wsdl:input>
      <wsdl:output>
        <soap:body use="literal"/>
      </wsdl:output>
    </wsdl:operation>
</wsdl:binding>
```

- **wsdl:service**: corresponde al servicio en sí mismo. El nombre del servicio y el puerto se utilizan para formar la URL con la que acceder al servicio. A esta URL también se le conoce como *endpoint*. Su formato es el siguiente:

http://host:puerto/axis2/services/nombre_portType.

```
  <wsdl:service name="editorial">
    <wsdl:port binding="tns:editorialSOAP" name="editorialSOAP">
      <soap:address location="http://www.example.org/"/>
    </wsdl:port>
  </wsdl:service>
```

Una vez definido el WSDL correspondiente al servicio Web, se invoca al método WSDL2Java, incluido en el *plug-in* de Eclipse Code Generator Wizard, que se encarga de interpretar el fichero y construir el esqueleto del servicio Web. De este *plug-in* se hablará con más detalle en el capítulo siguiente.

7.6.3. El servicio Web StockUpdate

Tras construir el archivo WSDL es posible generar el servicio a partir de él. Se crearán una serie de clases que conformarán el servicio Web. La estructura de directorios y archivos es la siguiente:

- **src**
  - **es.pfc.editorial.axis2.ws**
    - EditorialMessageReceiverInOut.java
    - EditorialSkeleton.java
  - **es.pfc.editorial.axis2.ws.messages**
    - ExtensionMapper.java
    - UpdateStockRequest.java
    - UpdateStockResponse.java
  - **es.pfc.editorial.axis2.ws.test**
    - UpdateStock.java
- **META-INF**
  - editorial.wsdl
  - editorialSchema.xsd
- **resources**
  - services.xml
- **build.xml**

Las clases *bean UpdateStockRequest* y *UpdateStockResponse* modelan la *request* y *response* del servicio Web.

La clase *UpdateStockRequest* contiene, entre otros, los métodos que asignan al mensaje SOAP el ISBN y el número de libros a actualizar. En la clase *UpdateStockResponse* hay métodos para asignar en el mensaje de vuelta el resultado de la operación. Además, ambas clases contienen gran parte de la lógica de intercambio de mensajes de Axis2. Uno de los aspectos más importantes es que aquí se encapsulan dos métodos para crear y *serializar* los *beans*, para lo cual implementan la interfaz *ADBBean*.

El método que crea y *serializa* los *beans* es el siguiente:

```
public javax.xml.stream.XMLStreamReader
        getPullParser(javax.xml.namespace.QName qName)
```

Este método permite obtener una representación XML del elemento que se le pasa como parámetro. Estos elementos pueden ser clases *bean* y tipos *complexType*.

```
public org.apache.axiom.om.OMElement getOMElement(
            final javax.xml.namespace.QName parentQName,
            final org.apache.axiom.om.OMFactory factory)
```

Como se puede ver en este método, se devuelve un objeto de tipo *OMElement*, que representa un objeto *ADBBean*. Dentro de este método se implementa un objeto de tipo *ADBDatasource*, que implementa el método `serialize()`, en el que se lleva a cabo la lógica de serialización para el *bean* en particular que se esté manejando.

La lógica de negocio se implementa en la clase *EditorialSkeleton*, que es la que contiene el método `updateStock()`. Este método recibe como parámetro un objeto de tipo *UpdateStockRequest*, del que obtiene el ISBN y la cantidad de los libros a actualizar. Actualiza el *stock* en la base de datos y apunta en la *response* un mensaje con el status de la operación. Este método devuelve un objeto *UpdateStockResponse*.

```
public UpdateStockResponse updateStock(UpdateStockRequest
                                                    updateStockRequest){
      UpdateStock objUpdate = new UpdateStock();
      UpdateStockResponse response = new UpdateStockResponse();

      response.setStatus(objUpdate.updateStock(updateStockRequest.getISBN(),
                          updateStockRequest.getCantidad()));
      return response;
    }
```

7.6.4. El cliente Web

Una vez que se ha descrito como está implementado el servicio Web, se mostrará la estructura del cliente que va a utilizar dicho servicio. Como ya se ha dicho, este cliente se corresponde con la editorial, que se conecta con el servicio Web y actualiza el *stock* en la base de datos. La estructura de directorios es la siguiente:

- **src**
  - **es.pfc.soap.editorial.clientws**
    - EditorialCallbackHandler.java
    - EditorialStub.java
    - EditorialClient.java
  - **es.pfc.soap.editorial.parser**
    - SAXParserEditorial.java
  - **es.pfc.soap.editorial.parser.bean**
    - StockUpdateBean.java
  - **es.pfc.soap.editorial.parser.documents**
    - stockUpdate.dtd
    - stockUpdate.xml
- **build.xml**

Esta estructura se divide en cuatro paquetes, el primero de ellos contiene las clases necesarias para la conexión contra el servicio Web y los otros tres contienen la lógica que interpreta un fichero XML y prepara el mensaje SOAP de envío en el lado del cliente.

El paquete *es.pfc.soap.editorial.clientws* contiene las siguientes clases:

- *EditorialCallbackHandler*: es una clase abstracta que representa el manejador de respuestas asíncronas. Esta clase se utiliza si se desean realizar llamadas asíncronas al servicio Web.
- *EditorialStub*: esta es la clase *stub* cliente con la que se pueden invocar las operaciones del servicio Web. Contiene un método por cada operación del servicio y clases internas que representan los elementos de los mensajes XML.
- *EditorialClient*: es la clase que contiene el método `main`. Crea los mensajes de petición y utiliza una instancia del *stub* para invocar la operación correspondiente y procesar la respuesta.

El resto de paquetes corresponden al *parser* de ficheros XML que se ha implementado para el servicio. Se ha definido que sea un fichero XML el que se cree desde la editorial con los datos del *stock* a actualizar. Este fichero XML está definido por el siguiente DTD:

```
<!DOCTYPE stock [
  <!ELEMENT stock (update+)>
```

```
  <!ELEMENT update (ISBN,cantidad)>
  <!ELEMENT ISBN         (#PCDATA)>
  <!ELEMENT cantidad     (#PCDATA)>
]>
```

Como se puede ver, contiene un elemento *stock* que a su vez contiene uno o más elementos *update*. Estos elementos *update* contienen un par de elementos que son el ISBN del libro a actualizar y la nueva cantidad disponible. A continuación se muestra un ejemplo de fichero XML:

```
<stock>
  <update>
      <ISBN>8401335744</ISBN>
      <cantidad>5</cantidad>
  </update>
</stock>
```

La clase *SAXParserEditorial* interpreta el fichero XML y crea un objeto de tipo *StockUpdateBean* por cada elemento *update* del fichero. Para *parsear* el fichero se ha utilizado el API SAX de Java, que permite tratar ficheros XML. Una vez interpretado todo el fichero se crea una lista que contiene todos los objetos *StockUpdateBean,* a partir de los cuales la clase *EditorialClient* creará las *request* que enviará al servicio Web. A continuación se ve cómo se ha implementado esta lógica:

```
String sEndPoint = "http://localhost:8880/axis2/services/editorial";
EditorialStub stub = new EditorialStub(sEndPoint);
System.out.println("*** Stub creado para endpoint = " + sEndPoint);

SAXParserEditorial sp = new SAXParserEditorial(pathDocument);
ArrayList<StockUpdateBean> listaStock = sp.getElementList();
for (int i = 0;i<listaStock.size();i++){
  StockUpdateBean s = listaStock.get(i);
  UpdateStockRequest updateStockRequest = new UpdateStockRequest();
  updateStockRequest.setISBN(s.getISBN());
  updateStockRequest.setCantidad(s.getCantidad());
  UpdateStockResponse updateStockResponse =
                              stub.updateStock(updateStockRequest);
  System.out.println("*** Invocando operacion update " +
                              updateStockResponse.getStatus());
}
```

Como se puede ver, primero se crea una instancia del *stub* indicándole la URL del *endpoint* del servicio. Después se invoca al *parser* SAX para obtener la lista de *beans* con los datos del fichero. Por cada uno de estos *bean* se crea una *request* a la que se pasa como parámetros el ISBN y la cantidad. Se invoca el método `updateStock` del *stub* instanciado y se obtiene la *response* con el resultado de la operación.

7.7. Ejemplos de funcionamiento

A continuación se muestran dos ejemplos del funcionamiento de la aplicación Web y se describen las acciones que se llevan a cabo internamente desde que se pulsa una opción en una vista hasta que se devuelve la siguiente vista al usuario.

7.7.1. Acceso a la aplicación

Los siguientes pasos son necesarios para mostrar la página de Novedades cuando el usuario accede a la aplicación.

1. Cuando el usuario accede por primera vez a la aplicación se le muestra siempre la página de Novedades, puesto que es la redirección por defecto definida en el fichero *struts-config.xml*.

```
<global-forwards>
<forward name="welcome"
         path="/Welcome.do?accion=iniciarAplicacion"/>
</global-forwards>
```

2. A continuación se resuelve la página a mostrar definida en el mismo fichero.

```
<action-mappings>
  <action path="/Welcome"
              type="es.pfc.shop.actions.ActionInitApp"
              name="form_tabs"
              parameter="accion">
<forward name="fwd_mostrar_novedades" path="page.novedades"/>
</action>
```

Cuando se invoca la acción `/Welcome`, al estar definida de tipo `es.pfc.shop.actions.ActionInitApp`, el motor de navegación se posiciona en dicha clase. La acción definida en el paso 1 es `accion=iniciarAplicacion`, por lo que dentro de esta clase se va a buscar el método con este nombre.

```
public ActionForward iniciarAplicacion (ActionMapping mapping,
                            ActionForm form,
                            HttpServletRequest request,
                            HttpServletResponse response) throws Exception{

    return mapping.findForward("fwd_mostrar_novedades");
}
```

3. Este método devuelve la siguiente transición a ejecutar, `fwd_mostrar_novedades`. Como se ve en el fichero *struts-config.xml*, cuando se recibe esta transición la siguiente página a mostrar es `page.novedades`.

```
<forward name="fwd_mostrar_novedades" path="page.novedades"/>
```

4. Al tener dado de alta el uso de Tiles en la aplicación, el motor de navegación va a consultar el fichero *tiles-defs.xml*. En este fichero busca la definición `page.novedades`:

```
<definition name="page.novedades" extends="base.definition">
    <put name="titulo" value="Elephant Books – Novedades"></put>
    <put name="body" value="/pages/novedades.jsp"></put>
</definition>
```

En esta definición se especifican el título y el cuerpo de la página. Además se indica que extiende de la definición `base.definition`, que consta de lo siguiente:

```
<definition name="base.definition" path="/pages/plantilla.jsp">
      <put name="logo" value="/pages/logo.jsp"></put>
      <put name="tabs" value="/pages/tabs.jsp"></put>
      <put name="pie" value="/pages/pie.html"></put>
</definition>
```

Esta es la definición de la que heredan todas las demás. Se encarga de colocar dentro de la página *plantilla.jsp* el logotipo de la tienda, las pestañas y el pie de página.

5. Una vez compuesta la página de Novedades a partir del conjunto de páginas mencionado anteriormente, se muestra al usuario:

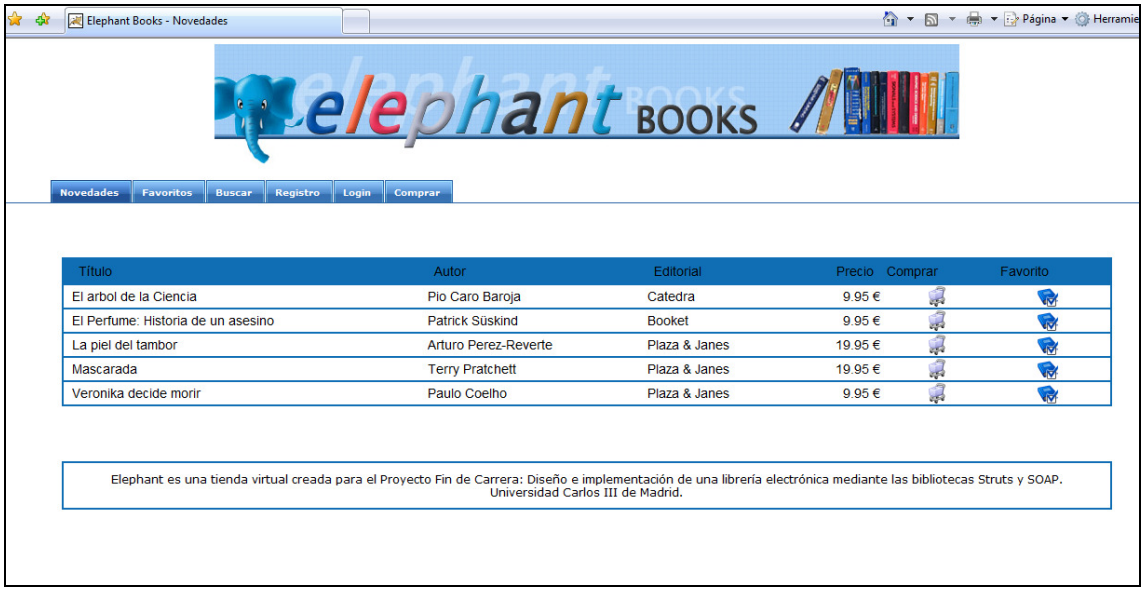


Figura 7-5. Pantalla de Favoritos.

7.7.2. Añadir un libro al carrito de la compra

A continuación se muestra qué eventos ocurren cuando se selecciona uno de los libros y se añade al carrito.

1. Se selecciona para su compra uno de los libros mostrados en la página de Novedades, pulsando sobre el icono. La página de Novedades está asociada al formulario `/novedades` definido al inicio de la página mediate la siguiente directiva:

```
<html:form action ="/novedades">
```

2. Al pulsar sobre el icono se dispara una función *javascript* que indica al formulario la acción a ejecutar.

```
document.forms[0].accion.value='event_add_item';
```

La function *javascript* también se encarga de hacer el *submit* del formulario. Una vez realizado, el motor de navegación de Struts consulta en el fichero *struts-config.xml* la clase `Action` asociada al formulario `/novedades`.

```
<action path="/novedades"
  type="es.pfc.shop.actions.ActionNovedades" name="form_novedades"
  parameter="accion">
  .
  .
  .
</action>
```

3. A continuación, el motor de navegación se posiciona en el método correspondiente a la acción asociada, en este caso `event_add_item` de la clase `Action` asociada al formulario, `es.pfc.shop.actions.ActionNovedades`.

```
public ActionForward event_add_item (ActionMapping mapping,
                                     ActionForm form,
                                     HttpServletRequest request,
                                     HttpServletResponse response)
                                  throws Exception{

  ActionForward f;

  FormNovedades frm = (FormNovedades)form;
  String idUsuario =
        (String)request.getSession().getAttribute(CTES.ID_USUARIO_ACTIVO);
  ArrayList lista = frm.getListaNovedades();
  Libro l = (Libro) lista.get(frm.getId_elemento_seleccionado());

  ArrayList listaCompra =
           (ArrayList)request.getSession().getAttribute(CTES.CART);
  if(listaCompra == null)
  listaCompra = new ArrayList();
  String ISBN = l.getISBN();
  listaCompra.add(ISBN);
  request.getSession().setAttribute(CTES.CART, listaCompra);

  return mapping.findForward(FWD_MOSTRAR_NOVEDADES);
}
```

En este método se ejecuta la lógica para guardar el libro a comprar en la sesión activa y se devuelve la siguiente transición a mostrar.

4. Para ver con qué página se corresponde la siguiente transición hay que consultar de nuevo el fichero *struts-config.xml*.

```
<action path="/novedades"
              type="es.pfc.shop.actions.ActionNovedades"
              name="form_novedades" parameter="accion">
      <forward name="fwd_mostrar_novedades" path="page.novedades"/>
      .
      .
      .
</action>
```

5. Al igual que en el caso anterior, se vuelve a mostrar la página *page.novedades*, que se construye mediante Tiles, como se ha explicado en el caso de uso anterior. En este momento en la aplicación se ha inicializado el carrito de la compra, se ha guardado el libro seleccionado en él y se ha almacenado en memoria.

7.8. Funcionamiento de la aplicación Web

7.8.1. Opciones del menú principal

En la parte superior se puede distinguir el menú principal, que consta de una serie de opciones que se describirán a continuación:

Figura 7-6. Menú de opciones de la tienda.

Novedades

Esta es la página que se muestra al iniciar la aplicación. Contiene el detalle de ciertos libros que están marcados como novedades en la base de datos. El usuario tiene la opción de comprar cualquiera de ellos o anotarlos como favoritos en el caso en que desee que estén disponibles más adelante.

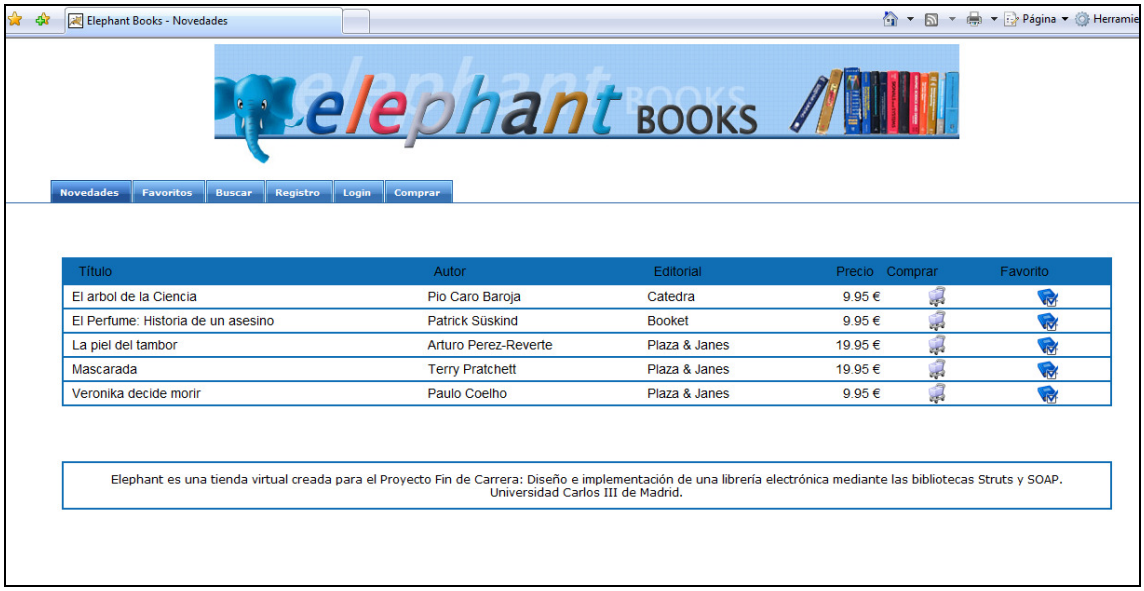


Figura 7-7. Pantalla de Novedades.

Favoritos

Desde esta opción se accede a la página de Favoritos. Si el usuario está autenticado se muestran los libros que tiene apuntados como favoritos, si los tuviera. Si el usuario no está validado en el sistema se le redirige a la página de Login.

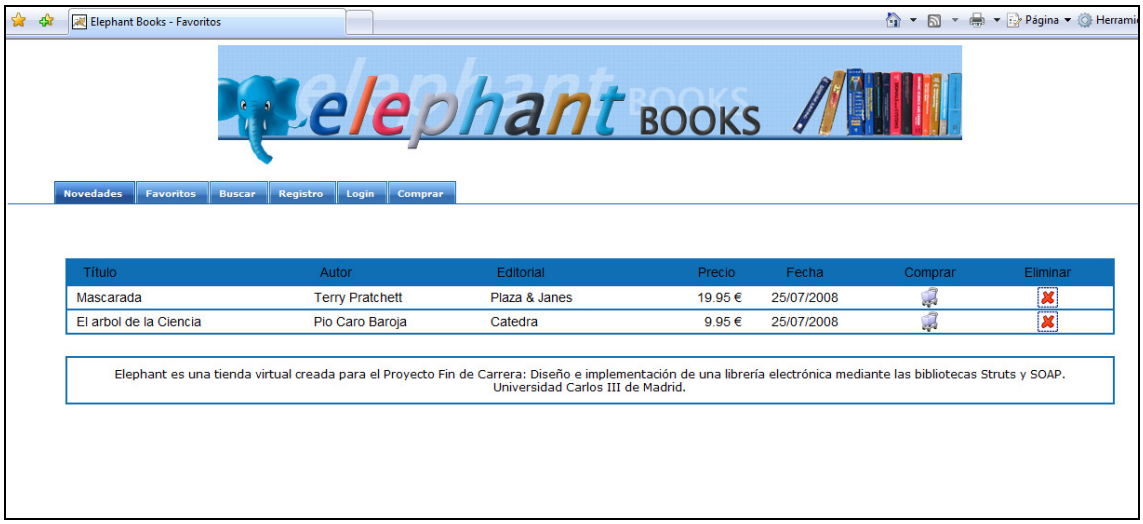


Figura 7-8. Pantalla de Favoritos.

Buscar

En la pestaña de búsquedas se da la opción de buscar un libro determinado atendiendo a diferentes criterios, como son por título, por autor o ambos.

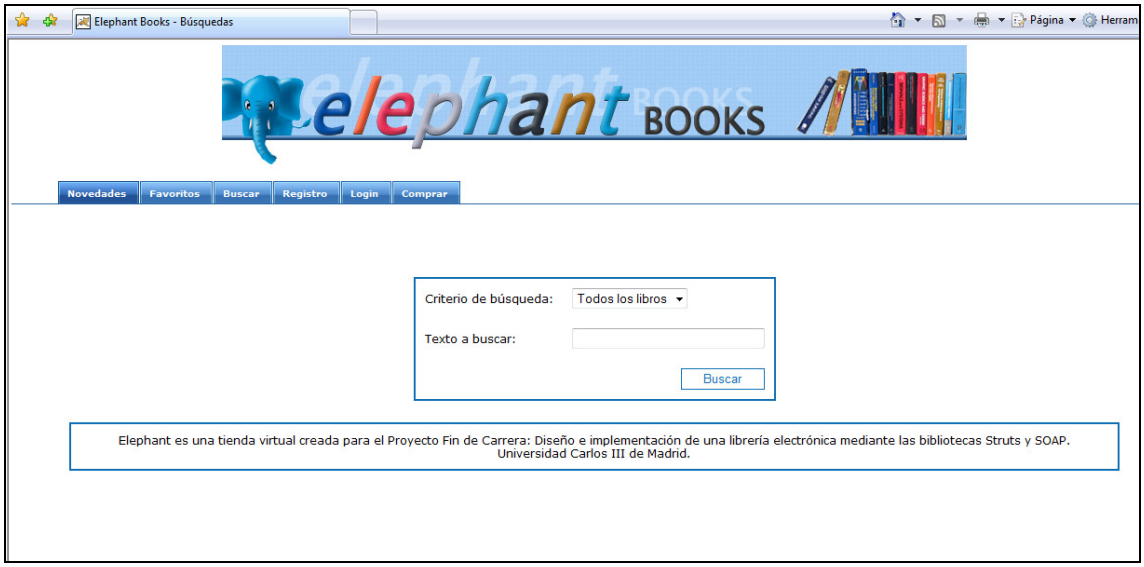


Figura 7-9. Filtro de Búsquedas.

Registro

Desde esta opción un usuario invitado se puede dar de alta para poder validarse como usuario autenticado y de esta forma poder efectuar compras o apuntar ciertos libros como favoritos.

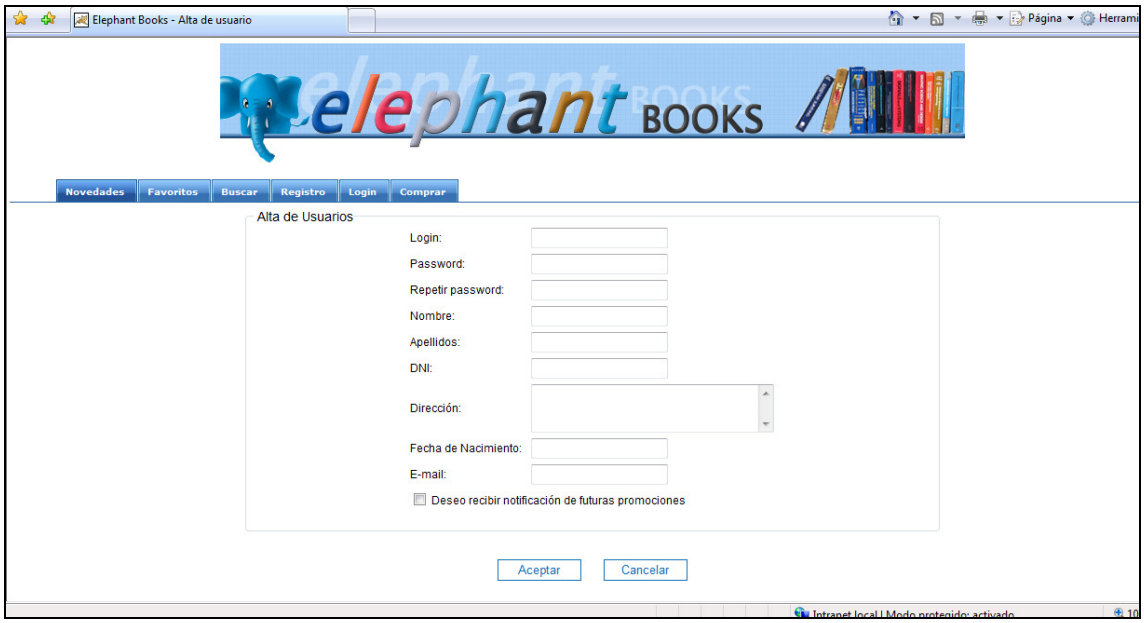


Figura 7-10. Pantalla de Alta de Usuarios.

Login

Muestra al usuario la página donde introducir sus credenciales para validarse como usuario registrado.

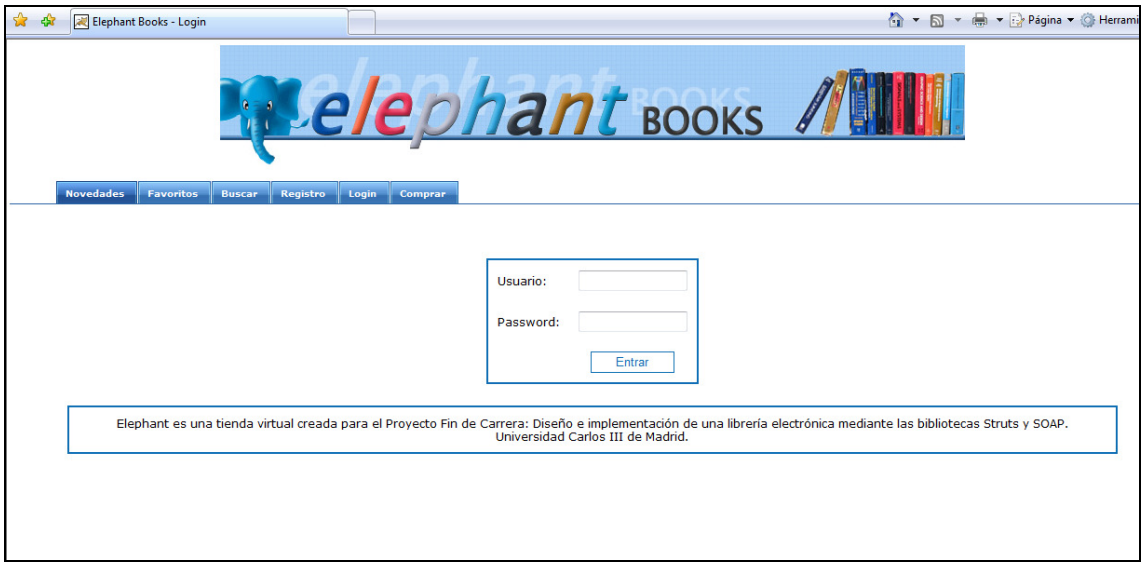


Figura 7-11. Pantalla de Login.

Comprar

En esta pantalla se muestra el carro de la compra del usuario y se ofrece la posibilidad de efectuar la compra o eliminar artículos de la lista. Para ver el carrito de la compra el usuario ha de estar autenticado en el sistema.

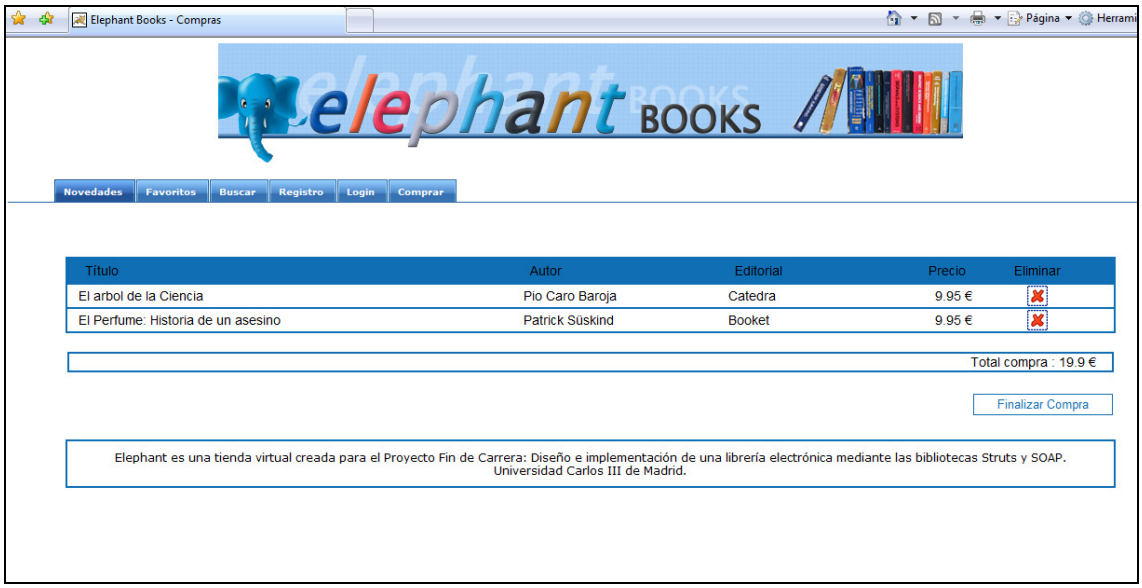


Figura 7-12. Pantalla de Compra.

7.8.2. Escenarios de uso en la aplicación Web

Usuario que realiza una compra

A continuación se va a ver paso a paso una navegación normal de un usuario que realiza una compra en la tienda.

1. El usuario accede a la tienda por primera vez y se le muestra la página de Favoritos.

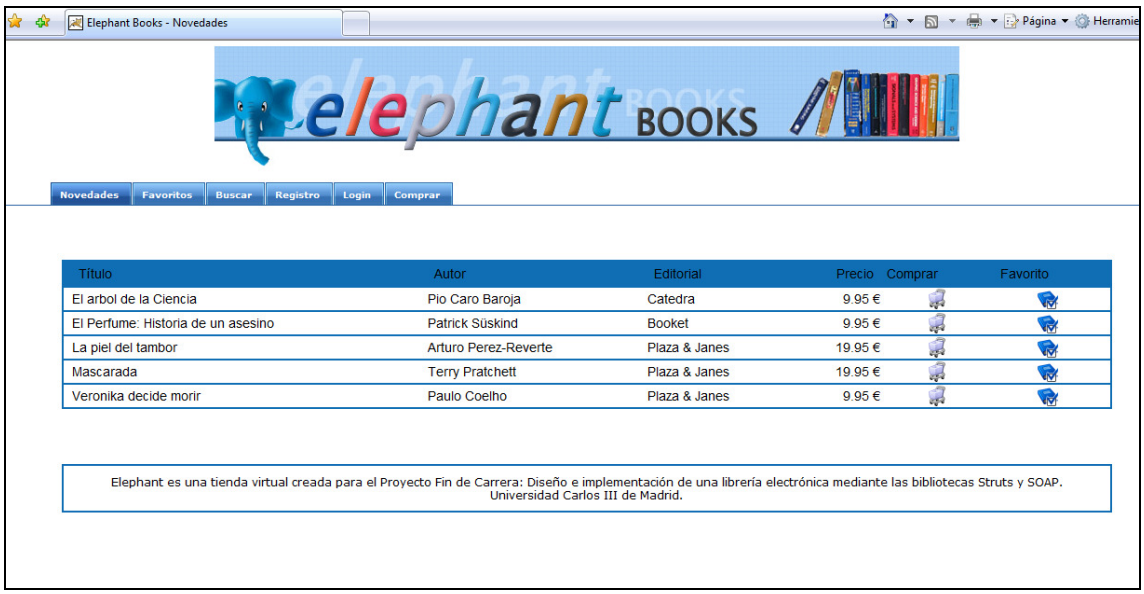


Figura 7-13. Página de Favoritos.

2. Se selecciona el libro que se desea comprar y se añade al carrito de la compra

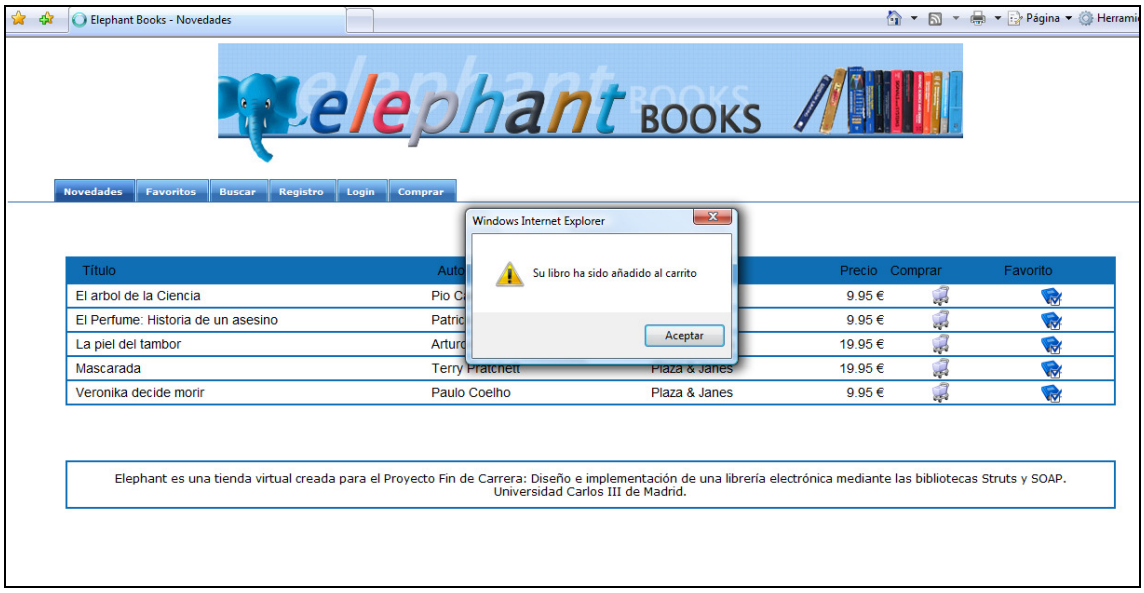


Figura 7-14. Carrito de la compra.

3. Si ya no desea comprar más libros, el usuario va a la pestaña *Comprar* para ver el estado de su compra

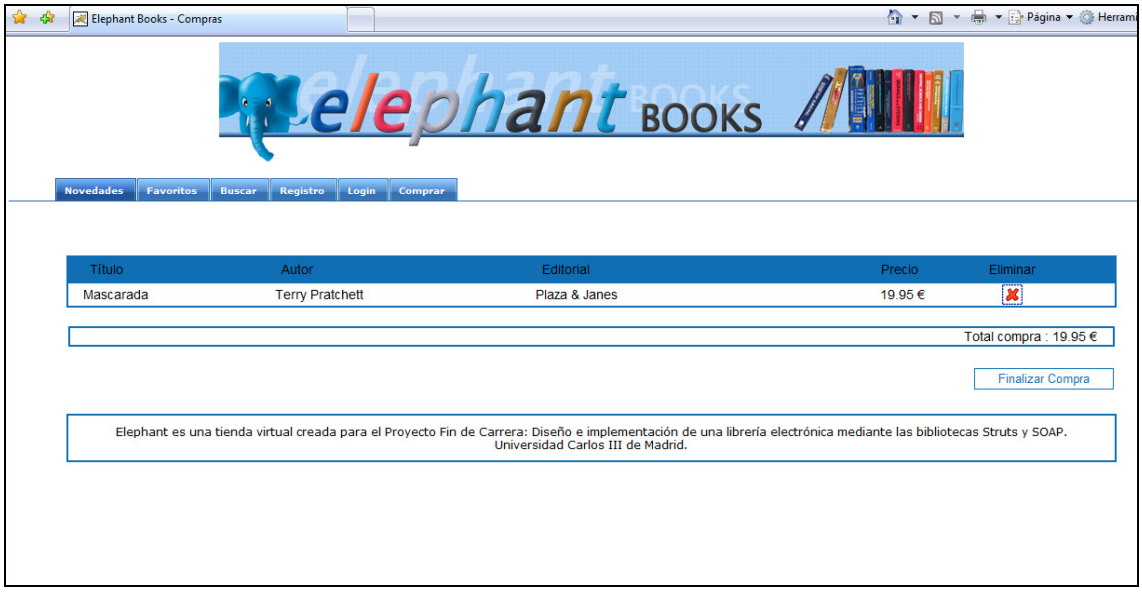


Figura 7-15. Página que muestra el estado de la compra.

4. Cuando el usuario decide finalizar la compra, pulsa el botón *Finalizar Compra* para efectuar el pago de la misma. El usuario no se ha validado previamente en el sistema, por lo que se le redirige a la página de Login.

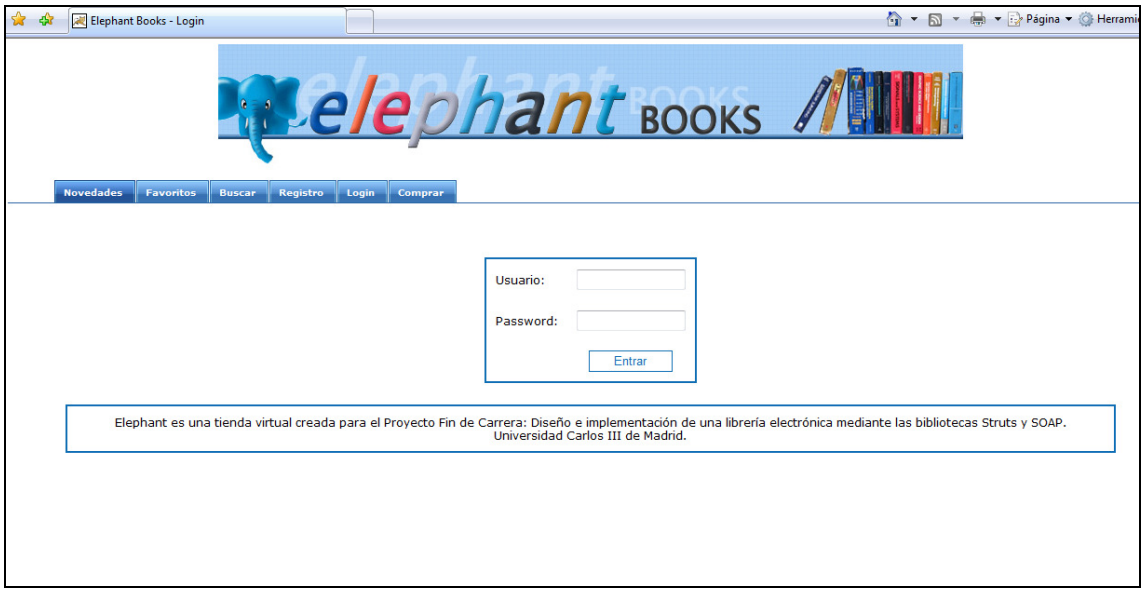


Figura 7-16. Página de Login.

5. Una vez que el usuario se ha validado, se le redirige a la página de compras para que valide su compra. Al pulsar sobre *Finalizar Compra* se muestran los datos personales del usuario y el detalle de la compra para que sea validado por el usuario.

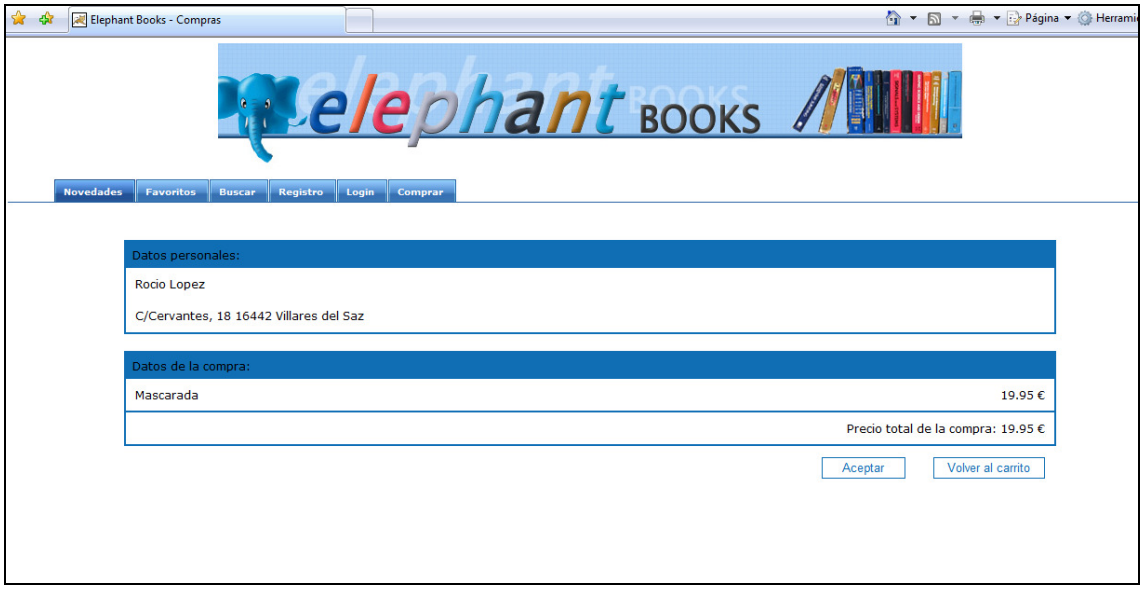


Figura 7-17. Validación de datos antes de la compra.

6. El usuario puede desde aquí volver al carrito de la compra o pulsar el botón *Aceptar* para continuar con la compra. En el caso que se muestra, se continúa con la compra. Se muestra la pantalla para introducir los datos bancarios.

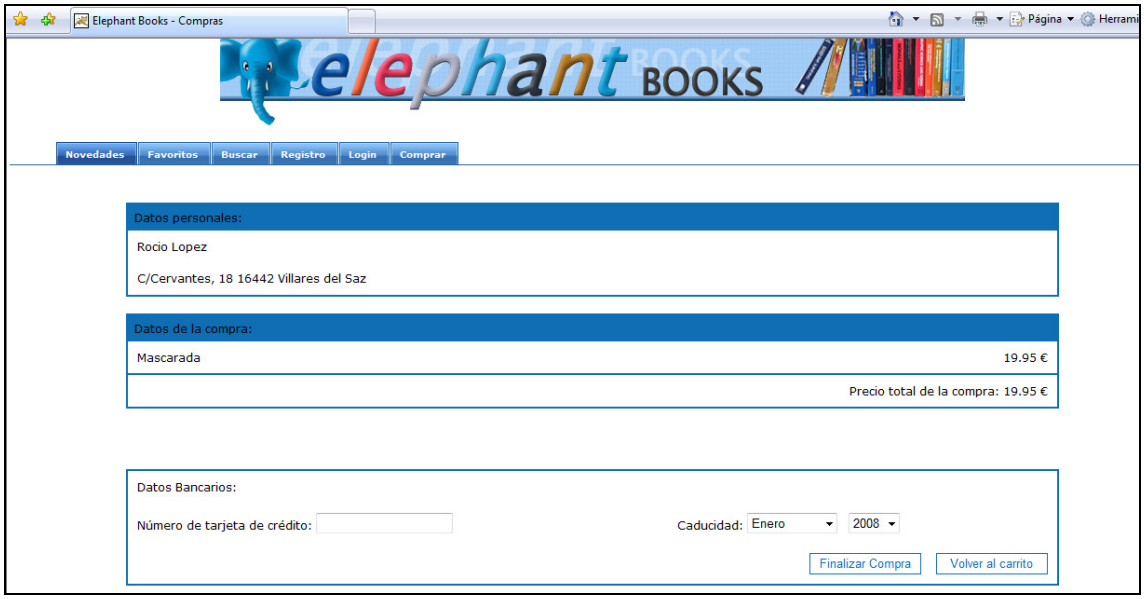


Figura 7-18. Pantalla de pago.

7. En esta pantalla el usuario introduce los datos de la tarjeta de crédito para efectuar el pago y pulsa el botón *Finalizar Compra*. Si los datos son correctos se almacena el pedido en la base de datos y se muestra la pantalla que indica que el proceso ha sido exitoso.

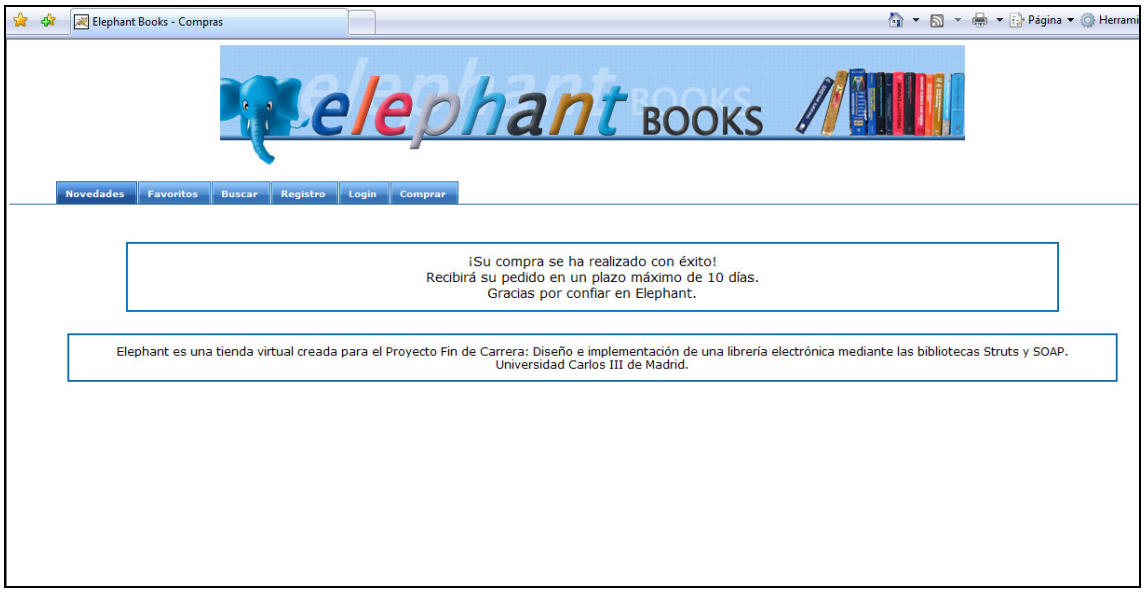


Figura 7-19. Pago confirmado.

Usuario que realiza una búsqueda y añade un libro como favorito

A continuación se muestra la navegación de un usuario que realiza una búsqueda y cuando encuentra el libro deseado lo añade como favorito para que esté disponible para ser comprado próximamente.

1. El usuario accede a la aplicación y se le muestra la página de Novedades.

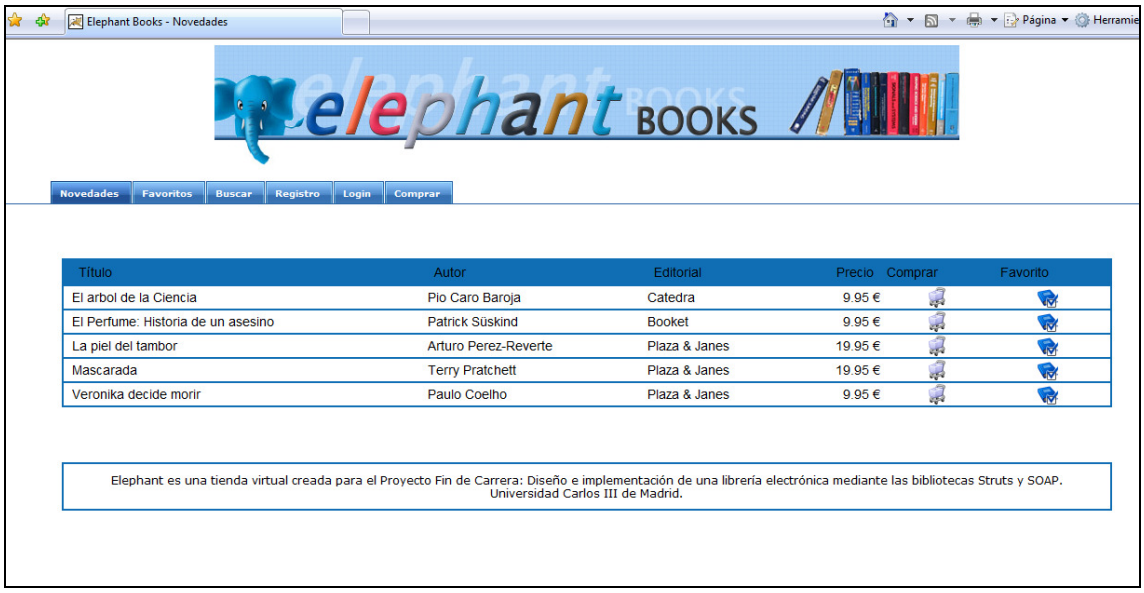


Figura 7-20. Página de Novedades.

2. El usuario se valida en el sistema mediante la pantalla de Login.

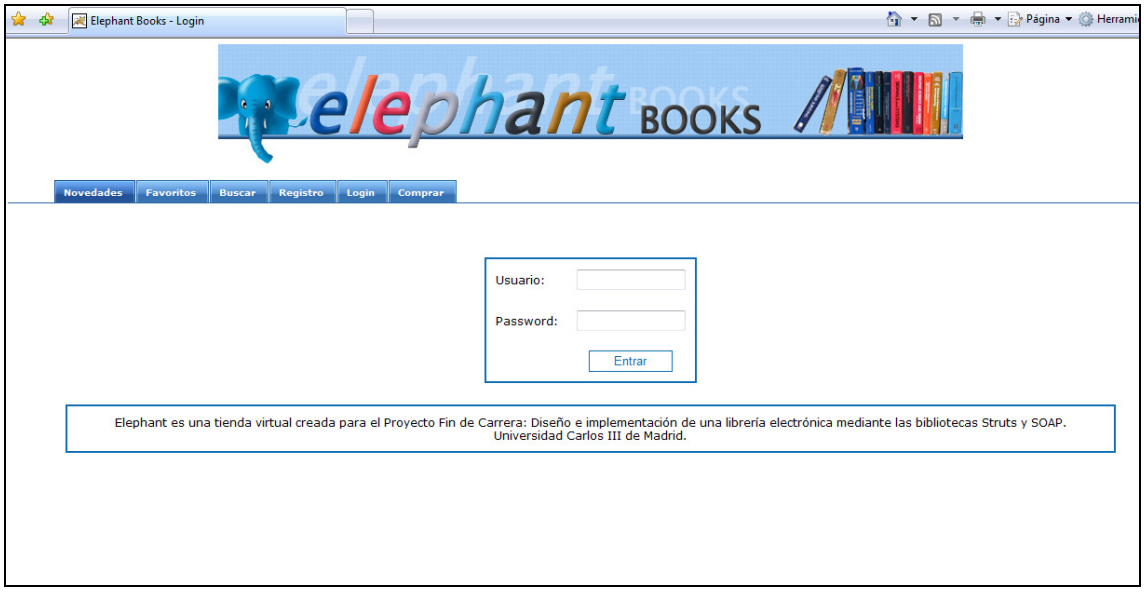


Figura 7-21. Página de Login.

3. El usuario desea buscar un libro concreto y pulsa la pestaña de *Buscar* para ver el formulario de búsquedas.

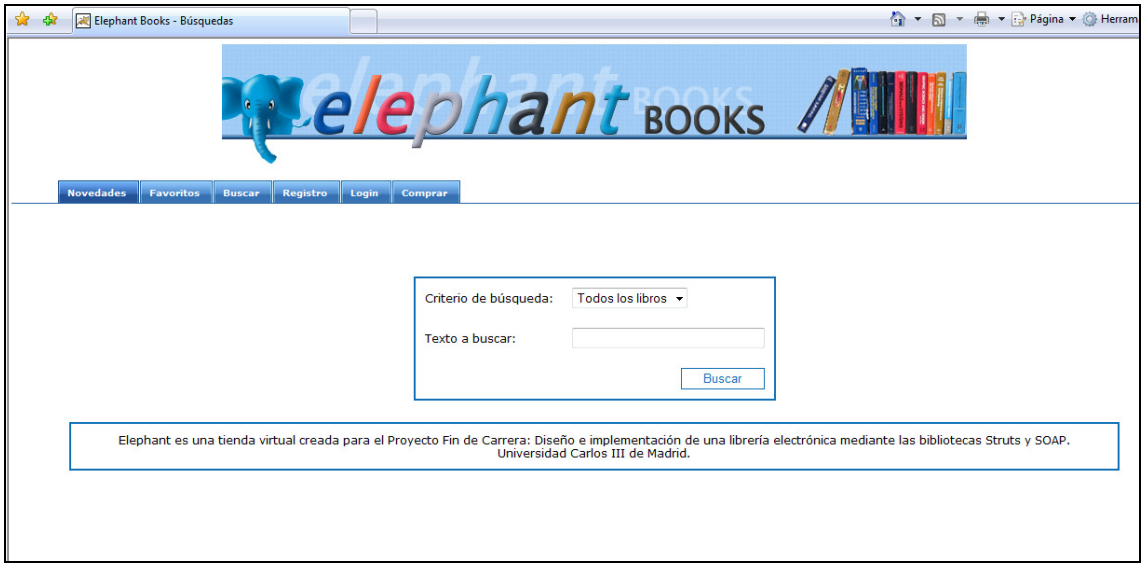


Figura 7-22. Página de Búsquedas.

4. El usuario introduce los parámetros de búsqueda y se le muestran los resultados obtenidos si los hubiera.

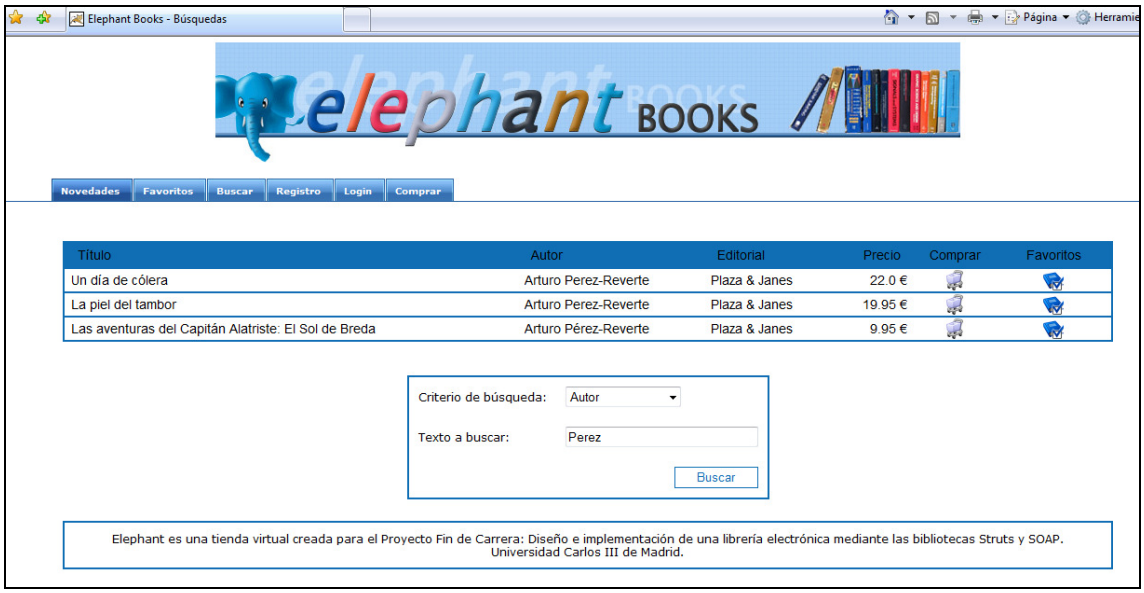


Figura 7-23. Resultados de la búsqueda.

5. El usuario desea apuntar uno de los libros como favorito para su posterior consulta y lo hace pulsando sobre el icono correspondiente.

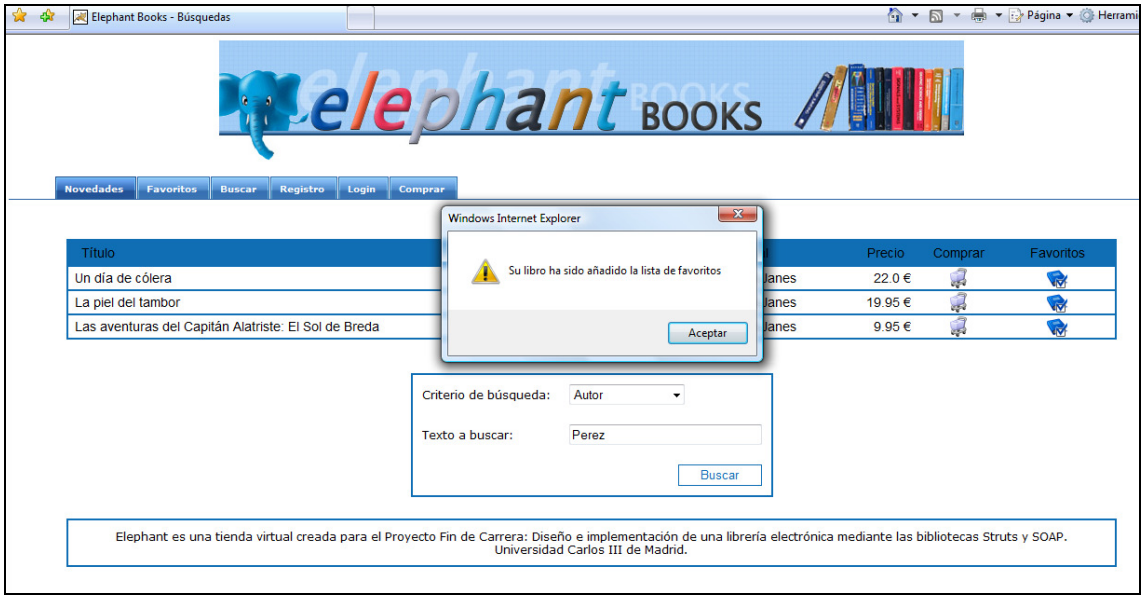


Figura 7-24. Favorito añadido.

6. El usuario consulta la lista de favoritos mediante la pestaña de Favoritos.

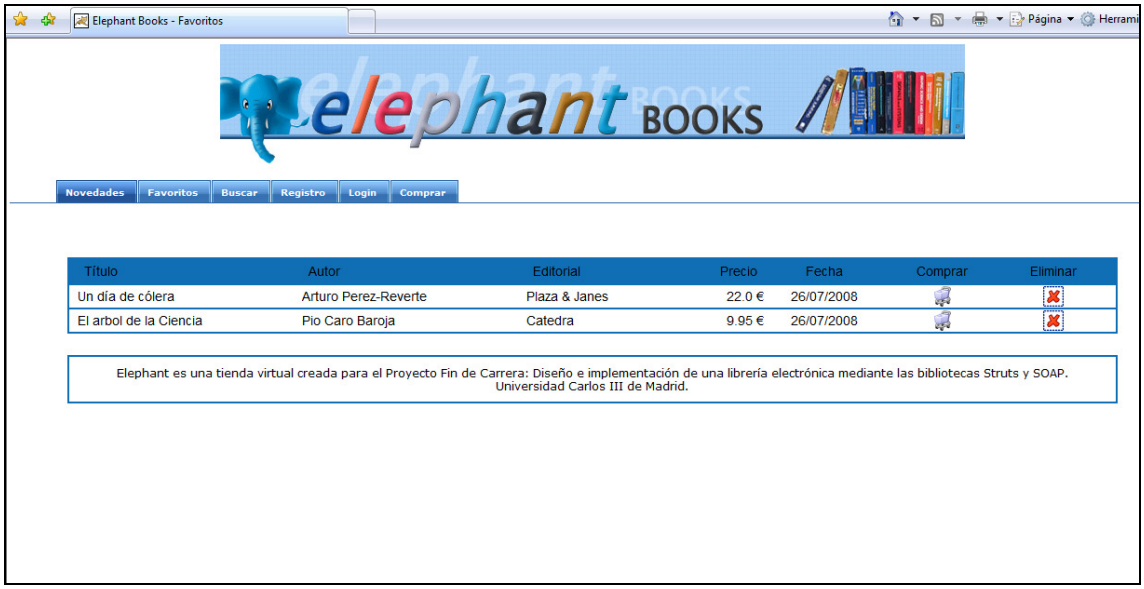


Figura 7-25. Pantalla de Favoritos.

7. El usuario decide posponer la compra y pulsa en la pestaña de Login para finalizar la sesión en el sistema.

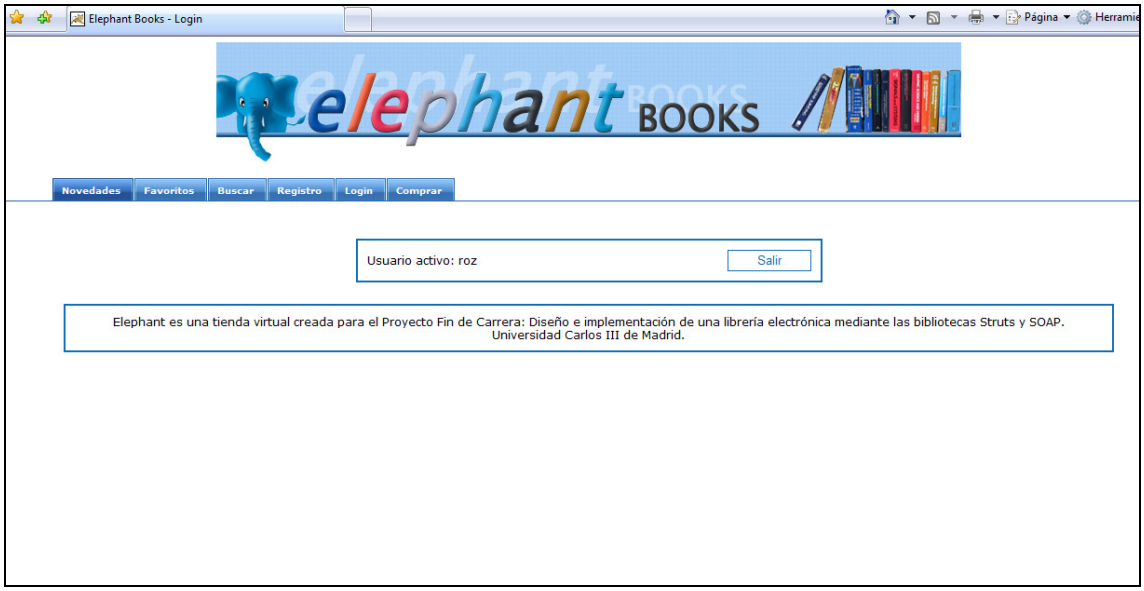


Figura 7-26. Página de Login con el usuario validado.

8. El usuario pulsa el botón *Salir* y se le redirige a la página de Novedades.

Usuario que se registra en el sistema

A continuación se muestran los pasos a seguir por un usuario que se da de alta en el sistema para poder acceder como usuario registrado.

1. El usuario accede a la aplicación y se le muestra la pantalla de Novedades.
2. El usuario accede a la página de registro pulsando la opción *Registro* del menú superior.

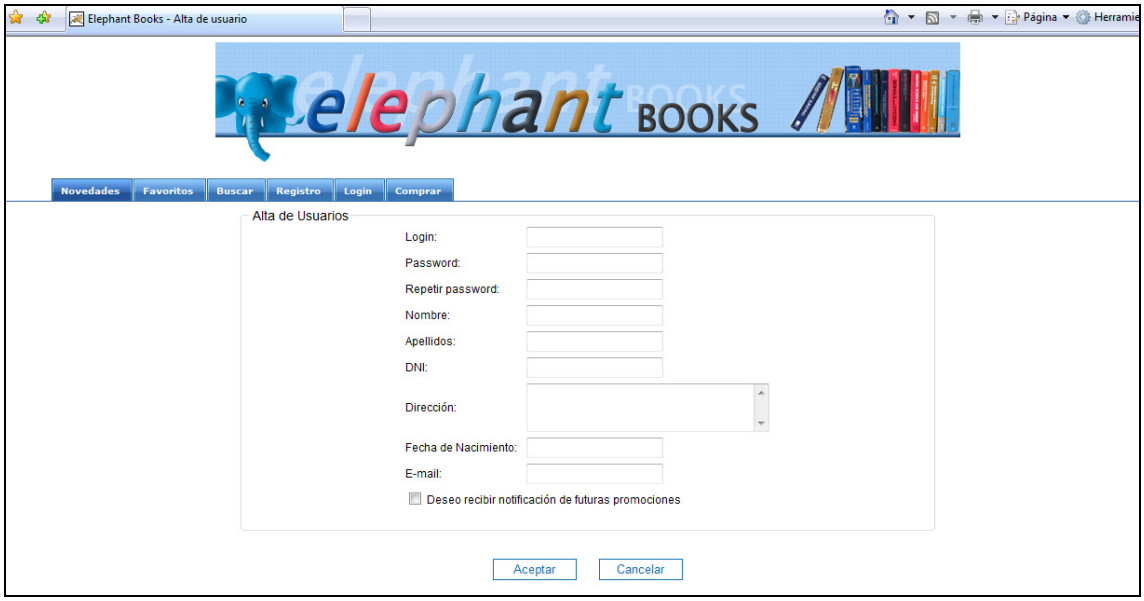


Figura 7-27. Alta de un usuario.

3. Si el usuario completa correctamente los datos y pulsa el botón *Aceptar*, éstos pasan a formar parte de la base de datos del sistema. Se le valida en el sistema y se le muestra la página de Novedades.

7.9. Funcionamiento de la aplicación SOAP

La aplicación SOAP permite actualizar el *stock* disponible de la tienda de una manera muy sencilla. El usuario editorial simplemente necesita crear un fichero XML con el ISBN que identifica al libro y la cantidad que desea añadir al *stock* actual. El fichero sería como el que se muestra a continuación:

```
<stock>
  <update>
      <ISBN>8401335744</ISBN>
      <cantidad>5</cantidad>
  </update>
  <update>
      <ISBN>8408073086</ISBN>
      <cantidad>5</cantidad>
  </update>
  <update>
      <ISBN>8422607239</ISBN>
      <cantidad>5</cantidad>
  </update>
</stock>
```

Una vez que se ha creado el fichero XML con la información que se desea actualizar en la base de datos se utilizará la clase *TestClient*, que es la que contiene el método `main`. Dicha clase interpreta el fichero XML e invoca al servicio Web pasándole el ISBN y la cantidad como parámetros de entrada.

El servicio Web invoca el método *stockUpdate* que se encarga de actualizar los datos en la base de datos.

8. Historia del proyecto

8.1. Planificación y presupuesto

Las tareas que se han seguido para la realización de este proyecto y la duración de las mismas son las que se muestran a continuación.

Id	Nombre de tarea	Duración
1	**PFC - Elephant**	**200 días**
2	**Elephant Aplicación Web**	**132 días**
3	Estudios previos	15 días
4	Instalación y configuración software	7 días
5	Análisis funcional de la aplicación	10 días
6	Diseño técnico de la aplicación	10 días
7	Diseño de la base de datos	5 días
8	**Implementación**	**49 días**
9	Página de Novedades	5 días
10	Página de Favoritos	7 días
11	Página de Búsquedas	9 días
12	Página de Registro	7 días
13	Página de Login	7 días
14	Página de Compras	14 días
15	**Configuración Tiles**	**21 días**
16	Fichero de configuración	3 días
17	**Diseño e implementación de plantillas**	**13 días**
18	Página Logo	2 días
19	Página Pestañas	7 días
20	Página Pie de página	2 días
21	Página plantilla	2 días
22	Hoja de estilos	5 días
23	Pruebas integradas	5 días
24	Resolución de incidencias	10 días
25	**Elephant Aplicación SOAP**	**26 días**
26	Instalación y configuración software	7 días
27	Implementación del servicio	10 días
28	Implentación del cliente	5 días
29	Pruebas	4 días
30	**Memoria**	**42 días**
31	Índice	10 días
32	Memoria	25 días
33	Revisiones	7 días

Como se puede ver, el grueso del trabajo se ha dedicado a la aplicación Web y el resto se ha dividido entre la aplicación SOAP y la redacción de la memoria.

Se han empleado 200 jornadas en la elaboración del proyecto. Si se estiman 8 horas por jornada, se obtiene un total de 1600 horas. El tutor de este proyecto ha dedicado también una media de 1,5 horas semanales al seguimiento del proyecto, lo que hace un total de 60 horas. Las tarifas aplicadas son las que marca el Colegio Oficial de Ingenieros Técnicos de Telecomunicaciones.

Además de los costes personales hay que añadir unos costes materiales para completar el presupuesto. A estos costes materiales se le aplica un coeficiente de amortización de 1/5.

Concepto	Horas	Precio	Total
Proyectista	1.600	60	96.000
Tutor	60	120	7.200
Acceso a Internet			300
Ordenador gama media		1.200	240
Microsoft Office 2007			60
Material de oficina y encuadernación			200
Base imponible			104.000
I.V.A (16%)			16.640
Total			120.640

8.2. Especificación de requisitos

Los requisitos que se definieron para el proyecto al comienzo del mismo han sido los siguientes:

Requisitos funcionales

- Se provee a la aplicación de una página a la que cualquier usuario puede acceder para darse de alta en el sistema y acceder a la aplicación para interactuar con ella.
- La aplicación tiene una página inicial en la que cualquier usuario registrado se puede validar, de forma que se le muestre la interfaz correspondiente a sus privilegios de usuario.
- Las editoriales cuentan con una interfaz que les proporcionará la posibilidad de enviar ficheros a la aplicación. Estos ficheros serán documentos XML que invocarán los métodos necesarios para la inserción o modificación de entradas en la base de datos.

Requisitos no funcionales

- Los accesos a la base de datos se hacen de forma atómica, de tal modo que si se produjera un fallo en la aplicación se evitarían inconsistencias en los datos.
- No se ponen restricciones al tamaño de la aplicación, si bien se intenta evitar componentes innecesarios que puedan aumentarlo en exceso.

Durante el desarrollo de las aplicaciones se han cumplido estos requisitos y se han identificado algunos más que se han descrito en el capítulo 6.

8.3. Historia temporal

La idea original de este proyecto consistió en el desarrollo de una aplicación Web mediante el marco de trabajo Struts. En primer lugar, se decidió el tipo de aplicación a realizar, una tienda de libros *on-line*. La aplicación tendría un interfaz simple e intuitivo y un modelo de datos sencillo para poder centrar la atención en el desarrollo con Struts y mostrar sus capacidades. Esta es la idea principal que quiere plasmar el proyecto.

Una vez decidido qué tipo de implementación se pretendía llevar a cabo se hizo una primera especificación en la que se definieron las características de la tienda electrónica y cómo se pensaba desarrollar cada una de sus partes.

A continuación se hizo un estudio en profundidad de las características de Struts, cómo utilizarlo y cómo hacer una buena implementación de la aplicación. Uno de los objetivos más importantes del proyecto fue comprender el funcionamiento del motor de navegación, ya que el desarrollador debe cambiar un poco la forma de abordar el diseño de las aplicaciones con Struts. Se pasa de un modelo de *servlets* en el que el desarrollador se encarga de propagar parámetros de unas páginas a otras a otro en el que el *framework* es el encargado de esa tarea.

Cuando se alcanza la fase en la que se conoce Struts y se tiene la habilidad suficiente se hace un anteproyecto en el que se define cómo se va a implementar la aplicación, de qué partes constará y cómo el *framework* va a cubrir esas funcionalidades. Se trató de utilizar todo lo que Struts ofrece.

Al realizar este anteproyecto y definir qué usuarios de acceso habría en la aplicación, se pensó que en lugar de añadir una figura de administrador, que aunque aportaba funcionalidad a la aplicación no permitía utilizar ninguna característica adicional del marco de trabajo, se iba a añadir la figura de la editorial. Este tipo de usuario iba a acceder a la aplicación desde un sistema externo y podía tener sus sistemas desarollados en cualquier lenguaje de programación, por ello se le dotó de una pequeña plataforma de acceso a los datos implementada para utilizar un protocolo de comunicación muy común en el entorno empresarial SOAP. De esta forma, el proyecto pasó a ser una implementación y desarrollo de una librería electrónica utilizando Struts y SOAP.

Se definió en este anteproyecto el plan de trabajo mostrado en el apartado 8.1 y se comenzó con el desarrollo. Primero se realizó íntegramente la aplicación Web y una vez terminada se comenzó con el desarrollo de la aplicación SOAP. Para esta parte también se tuvo que hacer un estudio del protocolo, además de una selección de la implementación a utilizar. Una vez que se decidió el empleo de Axis2, se comenzó con su instalación y con el desarrollo. Este fue otro de los hitos importantes del proyecto, aunque quizá el aprendizaje de esta segunda tecnología no resultó tan costoso.

Una vez finalizados los desarrollos se comenzó a escribir la memoria del proyecto. En primer lugar se definió un índice a partir del cual se empezó a redactar, que ha sido modificado a medida que se ha avanzado en la memoria.

9. Conclusiones

El objetivo principal, propuesto al comienzo de este documento, consistía en evaluar el *framework* Struts como herramienta para el desarrollo de aplicaciones Web. Para ello se ha realizado un primer estudio en el que se han identificado sus capacidades y se ha comprendido su funcionamiento. A continuación se han puesto en práctica estos conocimientos adquiridos mediante el desarrollo de una tienda electrónica en la que se ha tratado de explotar al máximo estas capacidades. A partir del trabajo realizado con esta implementación se han obtenido una serie de conclusiones.

El uso de Struts facilita considerablemente la implementación de aplicaciones Web. Aunque es cierto que la curva de aprendizaje inicial es relativamente elevada, especialmente si no hay experiencia en el uso de *frameworks* de presentación, una vez superado este obstáculo inicial se consigue que el desarrollo de aplicaciones se haga de forma muy rápida. Al estar tanto el modelo como la vista muy gestionados por Struts, el desarrollador puede centrarse completamente en la lógica de negocio. Los ficheros de configuración de la aplicación son muy sencillos e inteligibles, además de que algunos parámetros de configuración pueden reutilizarse para otras aplicaciones de similares características. Como nota negativa se ha observado que la rigidez del motor de navegación implica que haya mucho código redundado y que dependa de la habilidad del desarrollador el dotar a la aplicación de ciertos patrones de reutilización de código.

El otro objetivo propuesto al comienzo de este proyecto ha sido probar que el empleo de servicios Web consigue dotar a las aplicaciones de una gran potencia. Para ello se ha complementado la aplicación Web con otra aplicación basada en servicios Web, para lo que se ha utilizado la implementación Axis2. Al igual que en el caso anterior, hay que realizar un esfuerzo inicial en el aprendizaje de los servicios Web y del protocolo elegido, SOAP. Este esfuerzo será mayor si no se tienen conocimientos de lenguajes de marcado. El uso de Axis2 como motor de servicios Web hace que el desarrollo se haga de una forma muy sencilla a partir de un WSDL definido. La complejidad depende del diseño de un buen fichero WSDL y no de la implementación de los servicios.

Por otro lado, estas dos tecnologías se integran perfectamente entre ellas permitiendo realizar aplicaciones más complejas sin tener que realizar un *middleware* para su integración.

Con esto se ha visto que el desarrollo de servicios Web se puede realizar de una forma muy sencilla con las herramientas adecuadas.

El desarrollo de aplicaciones Web con funcionalidades avanzadas se puede realizar de una forma estructurada, sencilla y relativamente rápida mediante el empleo de *frameworks*, como Struts. El desarrollador debe aprender a manejarlos y centrarse, a continuación, en la funcionalidad de la aplicación, plasmada en la lógica de negocio, dejando el resto de las tareas al *framework* empleado.

10. Anexo: Herramientas auxiliares

10.1. Eclipse Europa SDK

Para el desarrollo del proyecto se ha utilizado el entorno de desarrollo Eclipse Europa. Eclipse es un entorno de desarrollo integrado de código abierto independiente de la plataforma. Eclipse Europa contiene, de forma integrada, el IDE (entorno de desarrollo integrado) de Java, Java Developement Toolkit, y el compilador.

Eclipse comenzó como un proyecto de IBM como sucesor de su familia de herramientas para Visual Age. Eclipse es ahora desarrollado por la Fundación Eclipse, una organización independiente sin ánimo de lucro que fomenta una comunidad de código abierto y un conjunto de productos complementarios, capacidades y servicios.

La instalación del entorno dentro del sistema operativo es muy sencilla, ya que basta con descargarlo y descomprimirlo en el sistema de ficheros. Se puede descargar desde *http://www.eclipse.org*. Una vez descomprimido, su estructura de carpetas es la siguiente:

- Eclipse
  - configuration
  - features
  - plugins
  - readme
  - workspace
  - eclipse.exe
  - eclipse.ini
  - .eclipseproduct

Mediante el fichero *eclipse.exe* se lanza el entorno de desarrollo, sin necesidad de configurar nada más. La apariencia del entorno es la siguiente:

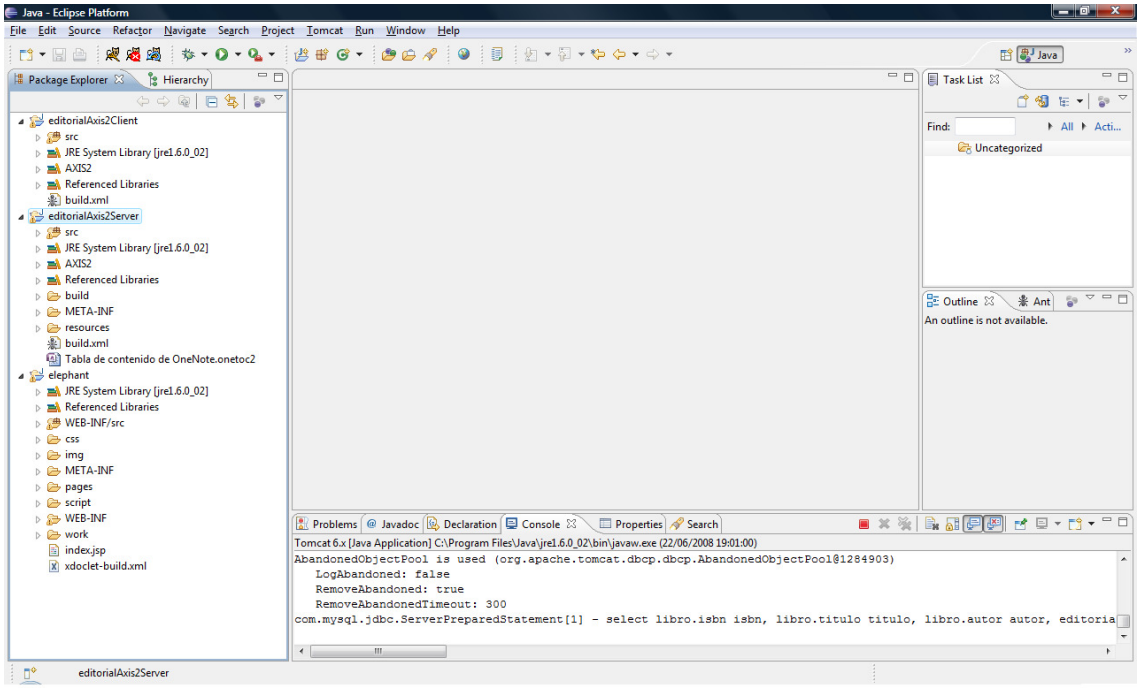


Figura 10-1. Entorno de desarrollo Eclipse.

El entorno de desarrollo integrado Eclipse emplea módulos o *plug-in* para proporcionar toda su funcionalidad a la plataforma. Cuando el usuario desea añadir una nueva funcionalidad, solamente debe añadir el *plug-in* apropiado al entorno. Esto contrasta con otros entornos monolíticos dónde todas las funcionalidades están incluidas, las necesite el usuario o no. Un *plug-in* es la unidad más pequeña que puede ser desarrollada y entregada por separado y, para que el entorno lo reconozca y lo despliegue, debe ir dentro de la carpeta *plugins*.

La carpeta *features* contiene los módulos correspondientes a las funcionalidades del entorno de desarrollo.

Dentro de la carpeta *configuration* se encuentran todos los ficheros de configuración necesarios para el entorno.

La carpeta *workspace* contiene uno o varios sistemas de ficheros donde se almacenan los proyectos del usuario. Cada proyecto contiene ficheros que son creados y manipulados por el usuario. Este acceso a los ficheros puede hacerse mediante Eclipse o cualquier explorador de ficheros del sistema operativo. Pueden crearse tantos *workspace* como el desarrollador considere y en cualquier parte del sistema operativo, no necesariamente dentro de la carpeta de Eclipse. Al iniciarse Eclipse es necesario indicar qué *workspace* se desea utilizar.

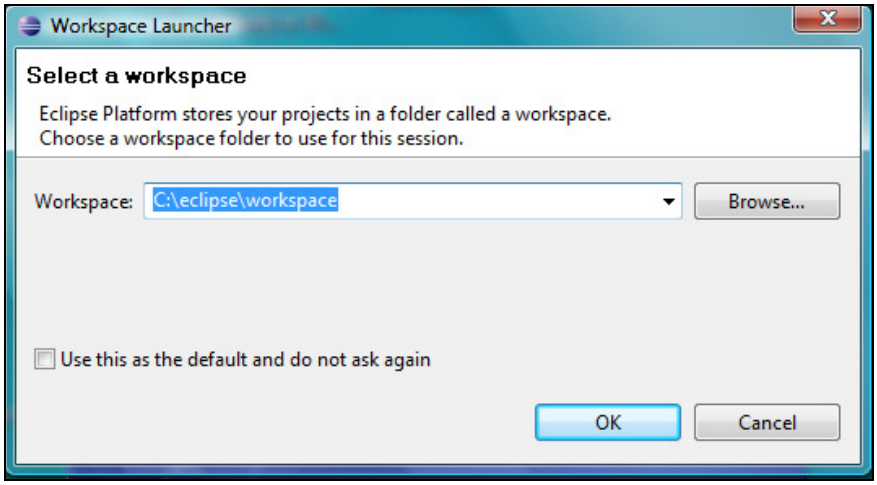


Figura 10-2. Selección de carpeta de trabajo en Eclipse.

Configuración del arranque

Cuando se arranca Eclipse se utiliza la máquina virtual de Java que encuentra en la variable PATH del sistema, aunque para los proyectos esté configurada otra. Si se quiere indicar a Eclipse qué máquina virtual de Java se debe utilizar en el arranque, es necesario configurar el fichero *eclipse.ini*, añadiendo la siguiente línea:

```
-vm c:/java/jre6/bin/javaw.exe
```

En este fichero también es posible configurar el tamaño inicial y el máximo de las memorias *heap* y *permgen*. La memoria *heap* es la asignada de forma dinámica y puede ser liberada por el recolector de basura. Para configurar los valores inicial y máximo se utilizan los parámetros de la máquina virtual `-Xms` y `-Xmx` respectivamente.

```
-vmargs -Xms128m -Xmx512m
```

La memoria *permgen* es memoria que se asigna permanentemente y no puede ser liberada. Se puede añadir en el fichero *eclipse.ini* la siguiente línea para gestionar dicho tamaño:

```
-launcher.XXMaxPermSize 256m
```

Además, también se pueden utilizar los parámetros `-XX:PermSize` y `-XX:MaxPermSize` de la máquina vitual para configurar los tamaños inicial y máximo de la memoria *permgen*. Por ejemplo:

```
-XX:PermSize=128m

XX:MaxPermSize=256m
```

10.2. Plug-in Tomcat para Eclipse Europa SDK

Como se ha visto, Eclipse completa su funcionalidad mediante *plug-ins*. Por ello se ha instalado para este proyecto un *plug-in* que permite controlar el servidor Tomcat desde el entorno de desarrollo. Esto resulta muy útil mientras se esta desarrollando, puesto que en muchas ocasiones es necesario reiniciar el servidor para que cargue cambios realizados.

Para instalar el *plug-in* simplemente hay que descargar el fichero *tomcatPluginv321.zip* y copiarlo dentro de la carpeta *plugins*. Al arrancar Eclipse, éste lo instalará y cargará. A partir de este momento aparecen tres nuevas opciones en la barra de herramientas de Eclipse, que permiten arrancar, parar y reiniciar Tomcat.

Figura 10-3. Opciones de control de Tomcat desde Eclipse.

Para que el entorno de desarrollo Eclipse conozca los parámetros de Tomcat es necesario configurarlo, para ello se pulsa en la opción *Window* → *Preferences*, y ahí en la pestaña de configuración de Tomcat:

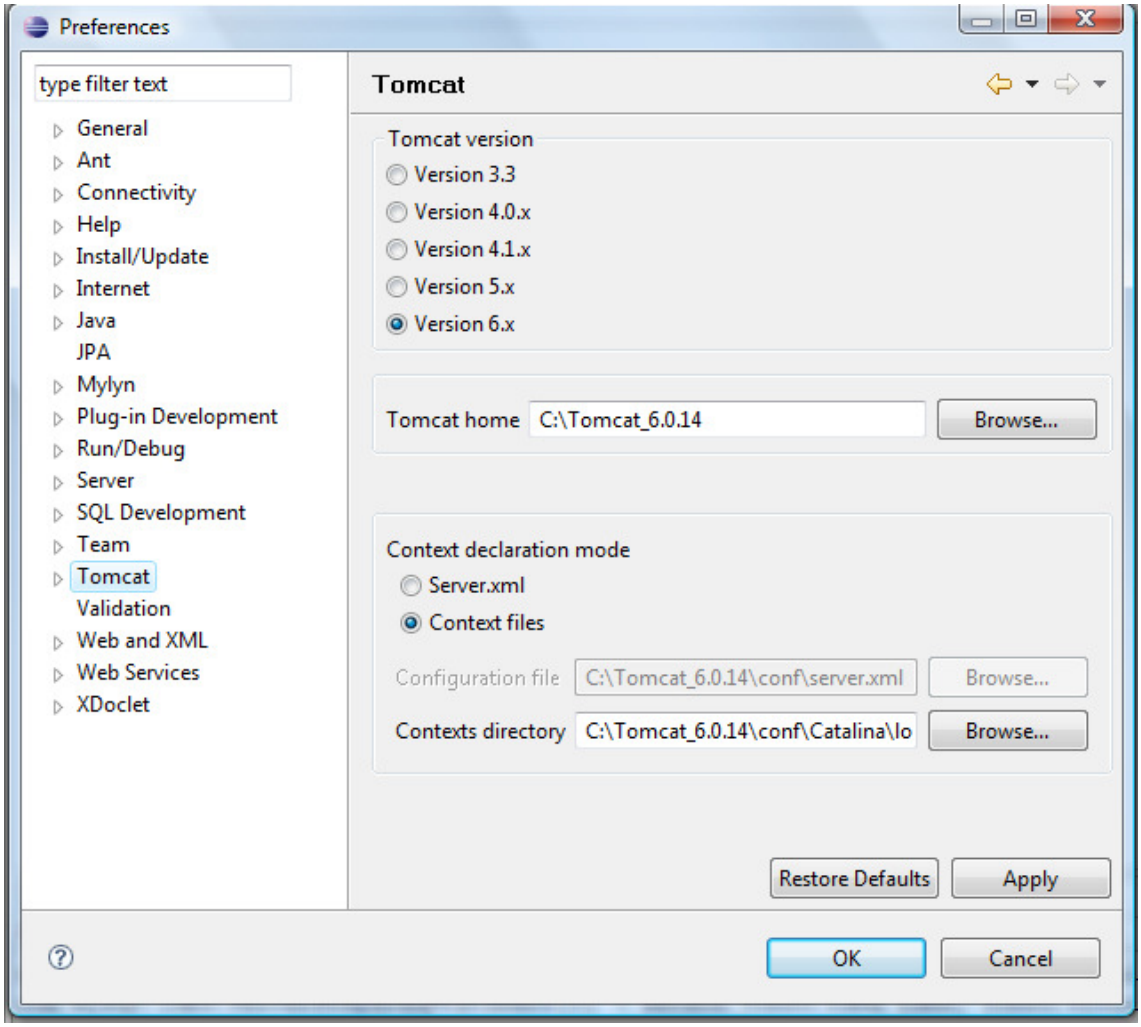


Figura 10-4. Configuración del *plug-in* Tomcat para Eclipse.

10.3. Code Generator Wizard para Eclipse Europa SDK

Para desarrollar el servicio Web y el cliente de SOAP se ha utilizado el *plug-in* para Eclipse *Code Generator Wizard*, de Axis2. Los *plug-ins* de Axis2 se pueden descargar desde la Web de Apache y para instalarlos solamente hay que descomprimirlos en la carpeta *plugins* de Eclipse.

Mediante este *plug-in* se puede generar el fichero WSDL desde una clase Java (Java2WSDL) o las clases Java desde el fichero WSDL (WSDL2Java).

Una vez descomprimido en la carpeta *plugins*, es necesario reiniciar Eclipse. Si se ha instalado correctamente, aparecerá un nuevo *wizard* en a la opción *File* → *New* → *Other*:

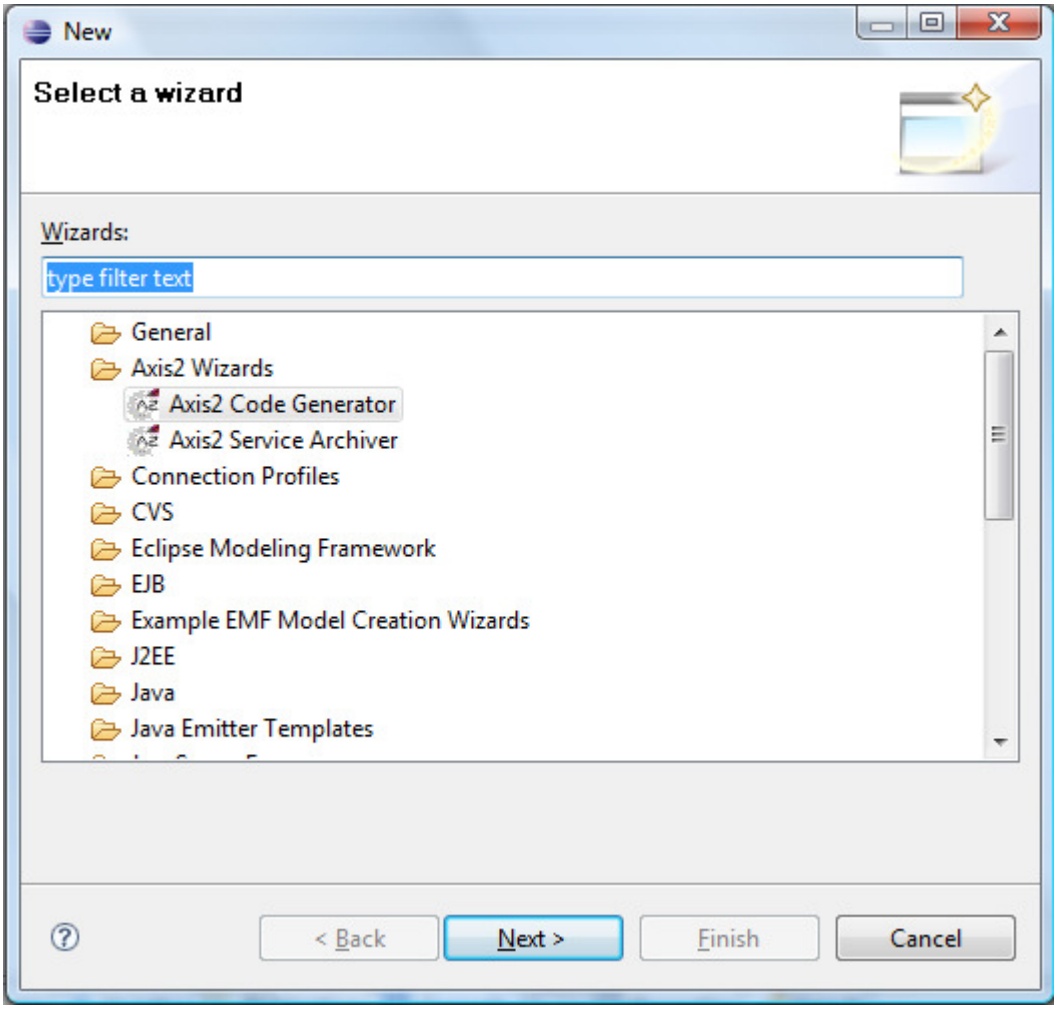


Figura 10-5. Code Generator Wizard. Selección del asistente.

Seleccionando *Axis2 Code Generator* y pulsando el botón *Next,* se iniciará el asistente:

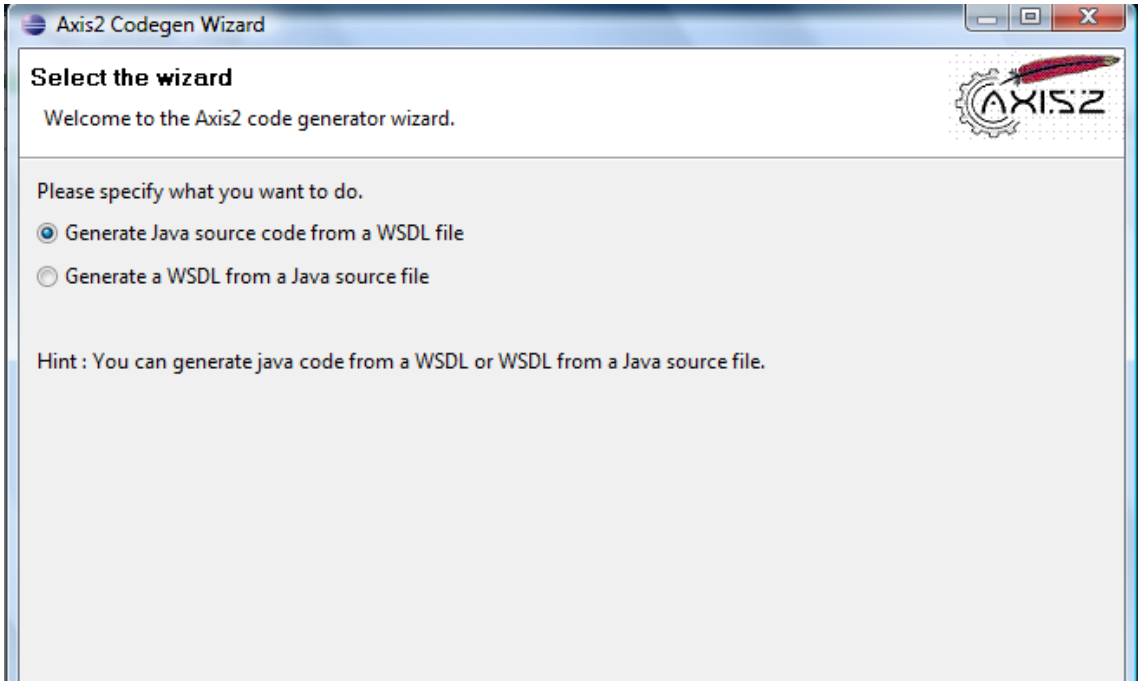


Figura 10-6. Code Generator Wizard. Selección de tipo de código a generar.

Como se puede ver, hay que seleccionar si se desea generar el código Java a partir del WSDL o generar el WSDL desde el código Java. Como ya se ha comentado, se elegirá la primera opción. Al pulsar el botón *Next*, pedirá la ruta del fichero WSDL:

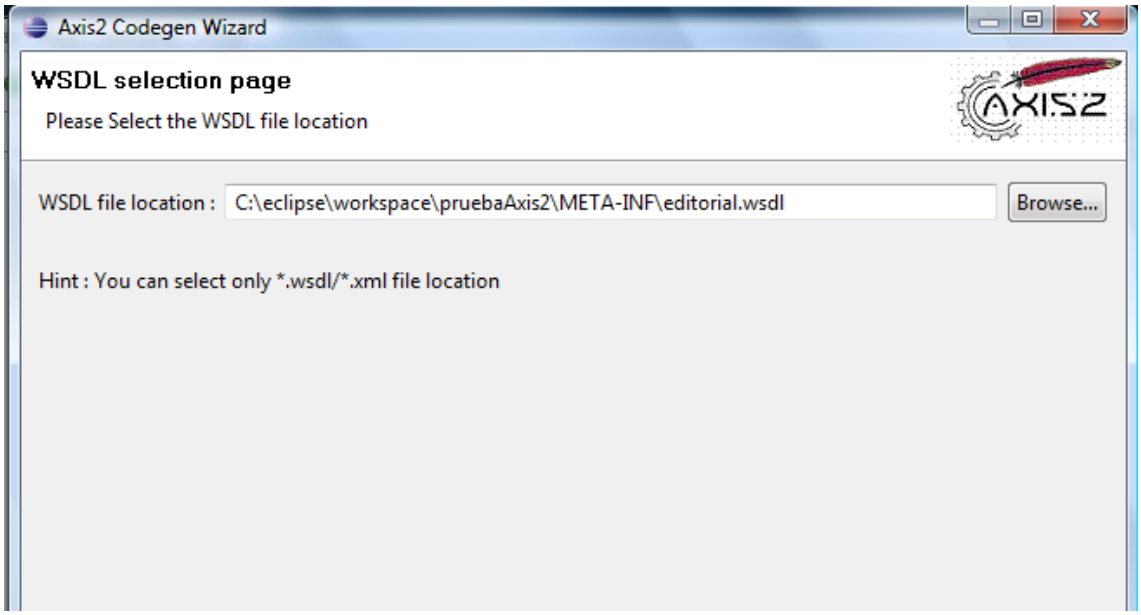


Figura 10-7. Code Generator Wizard. Selección de la ruta destino.

Se indicará la ruta del fichero y se pulsará el botón *Next*.

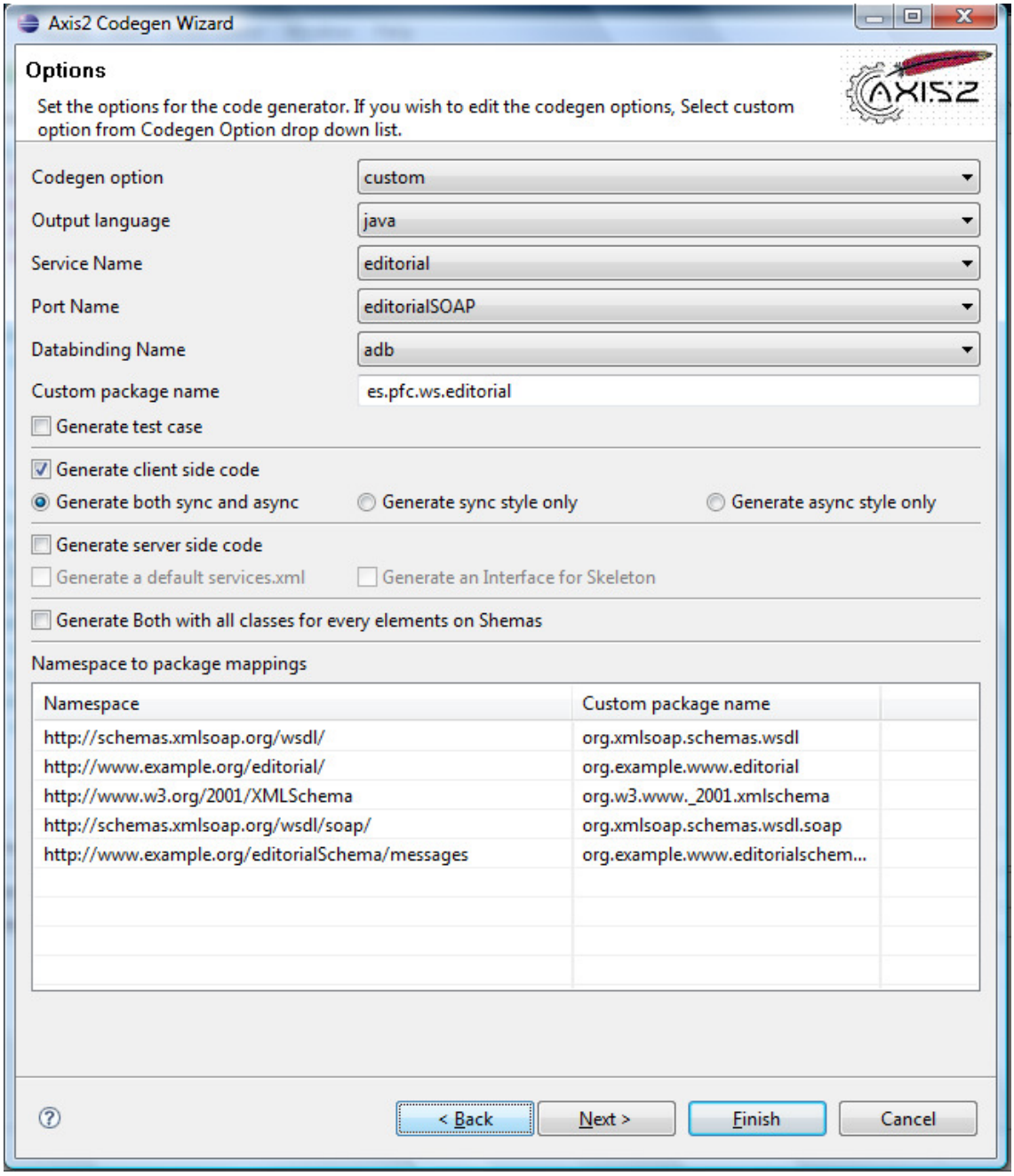


Figura 10-8. Code Generator Wizard. Opciones.

Aquí se seleccionan los parámetros para la generación de código. Se puede seleccionar si se desea generar el código del cliente o del servidor, si se quiere la generación de código para llamadas síncronas o asíncronas, o ambas, etc.

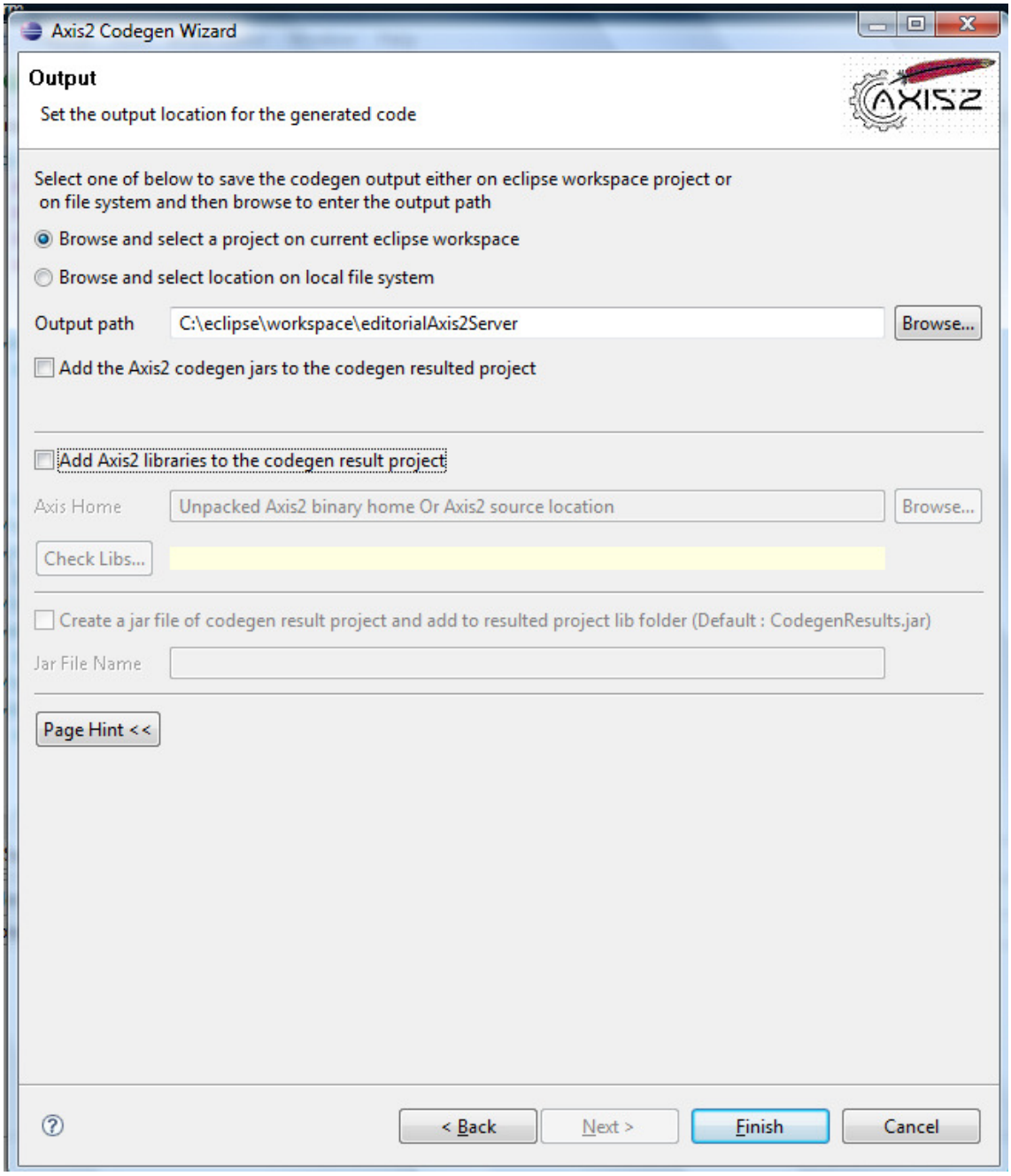


Figura 10-9. Code Generator Wizard. Salida.

Se selecciona el proyecto de salida y se pulsa el botón *Finish* para que genere las clases Java correspondientes.

10.4. MySQL GUI Tools 5.0

MySQL GUI Tools es un paquete de cuatro herramientas gráficas para facilitar el uso de MySQL. Dicho paquete se compone de las siguientes aplicaciones:

- **MySQL Administrator 1.2**: consola de administración gráfica del servidor de base de datos.
- **MySQL Query Browser 1.2**: para crear, ejecutar y optimizar consultas SQL.
- **MySQL Migration Toolkit 1.1**: para migrar esquemas y datos desde varias bases de datos relacionales como Oracle, Microsoft SQL Server o Microsoft Access a MySQL 5.0 o posterior.
- **MySQL Workbench 1.1**: herramienta gráfica para el diseño de bases de datos.

De este paquete de herramientas se han utilizado en el desarrollo del proyecto dos de ellas, las cuales se van a ver con más detalle a continuación.

10.4.1. MySQL Administrator

Al instalar una base de datos MySQL, la interfaz que se proporciona para gestionarlo es una interfaz de línea de órdenes. Este interfaz permite administrar la base de datos completamente, pero de una manera poco intuitiva. Para facilitar la gestión de la base de datos se ha instalado la utilidad MySQL Administrator, que consta de una interfaz gráfica muy completa diseñada para la administración de los servidores de base de datos MySQL.

A continuación se detallan algunas de las funcionalidades más importantes y útiles que ofrece MySQL Administrator.

Server Information

Al acceder a la aplicación, se selecciona automáticamente esta opción. En ella es posible ver información relacionada con la instancia del servidor de base de datos y con el cliente y servidor de base de datos. La información que ofrece es relativa a la IP, puerto, red, etc.

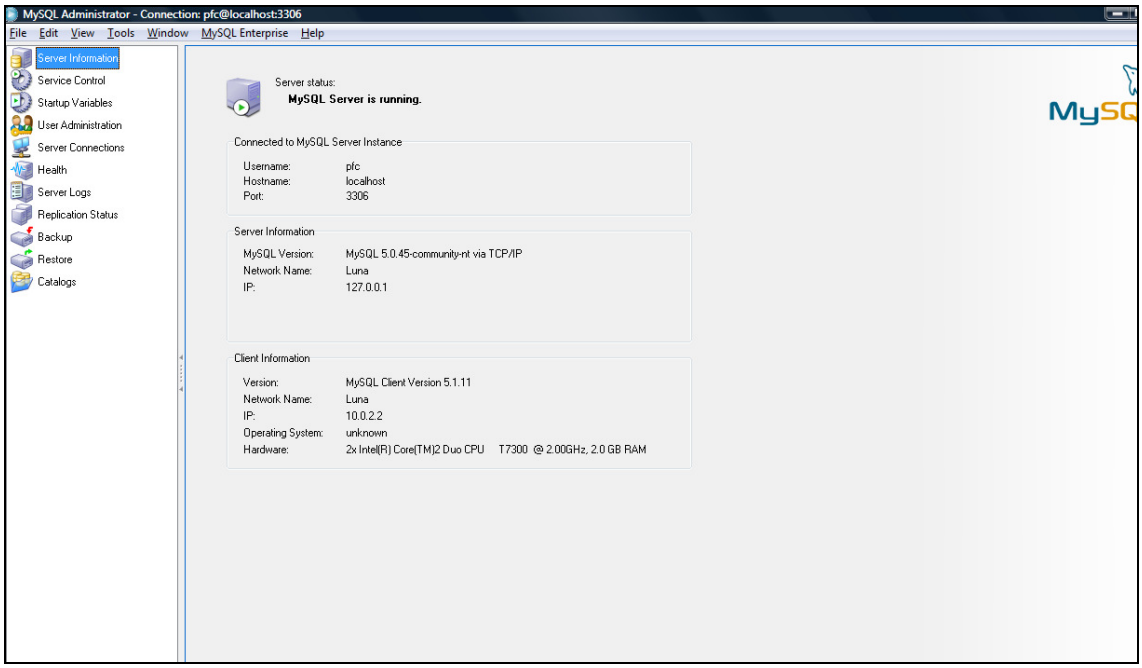


Figura 10-10. MySQL Administrator. Server Information.

Service Control

Desde esta interfaz se permite arrancar y parar el servicio y ver los correspondientes mensajes de *log*. Desde aquí se especifica el *path* del fichero de configuración que debe leer para obtener los parámetros del arranque.

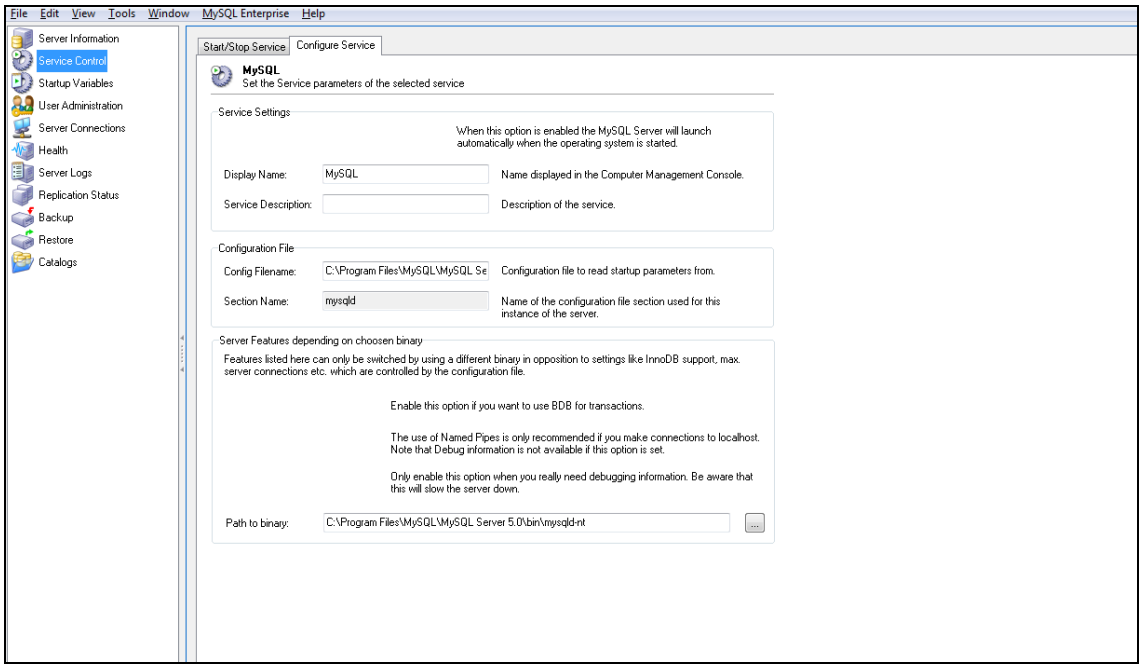


Figura 10-11. MySQL Administrator. Service Control.

Startup Variables

Desde esta pestaña se permite configurar las opciones de rendimiento del sistema. Permite especificar los tamaños de *buffer* de los índices e hilos y del tipo de almacenamiento por defecto. También permite hacer un *tunning* del sistema de base de datos en función de las necesidades de los esquemas y de las aplicaciones. Además, es posible definir el número máximo de conexiones simultáneas, número de conexiones por usuario, tamaño de la caché, etc.

También desde aquí se puede dar de alta otra base de datos MySQL en modo esclavo, de tal forma que haya redundancia de datos y se pueda, bien balancear la carga entre las dos bases de datos o tener otra disponible a la que se pueda atacar si se produce un fallo en la base de datos maestra.

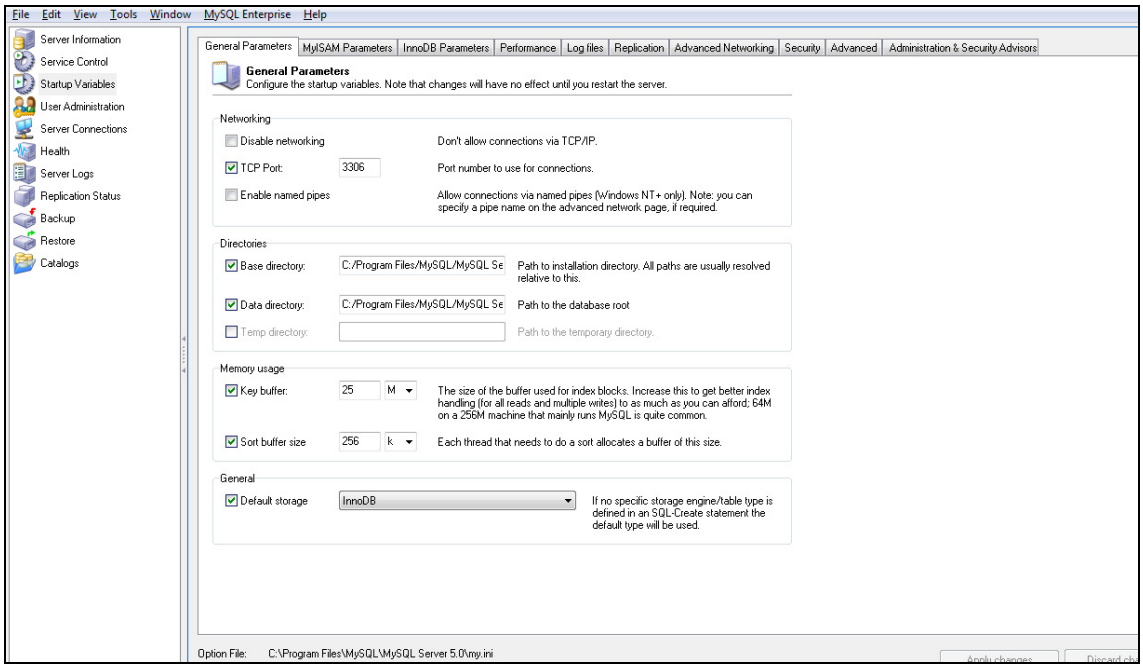


Figura 10-12. MySQL Administrator. Startup Variables.

User Administration

Permite crear distintos usuarios de acceso y otorgar privilegios sobre los esquemas a cada uno de ellos.

Server Connections

Desde aquí es posible conocer las conexiones que tiene establecidas la base de datos. Además, se puede ver el esquema que las ha originado, así como el tiempo que ha consumido cada conexión en la transacción que ha llevado a cabo.

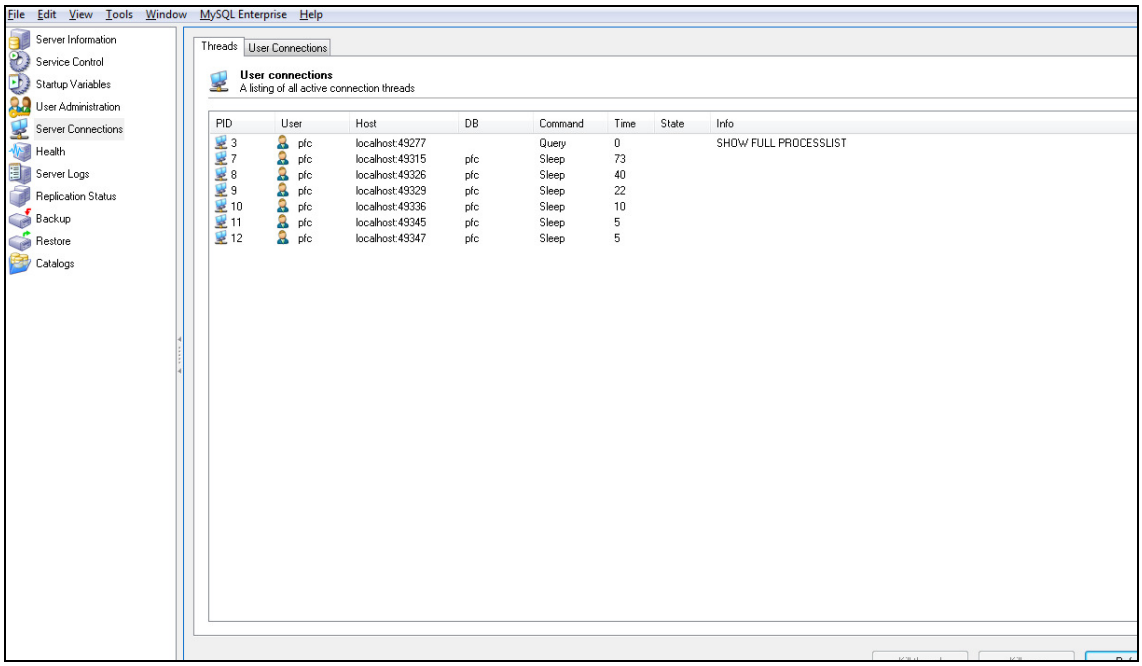


Figura 10-13. MySQL Administrator. Server Connections.

Health

Da información del estado de la memoria, del trafico enviado, *buffer* de memoria utilizado, etc. Además permite ver cuántas transacciones y de qué tipo se han llevado a cabo.

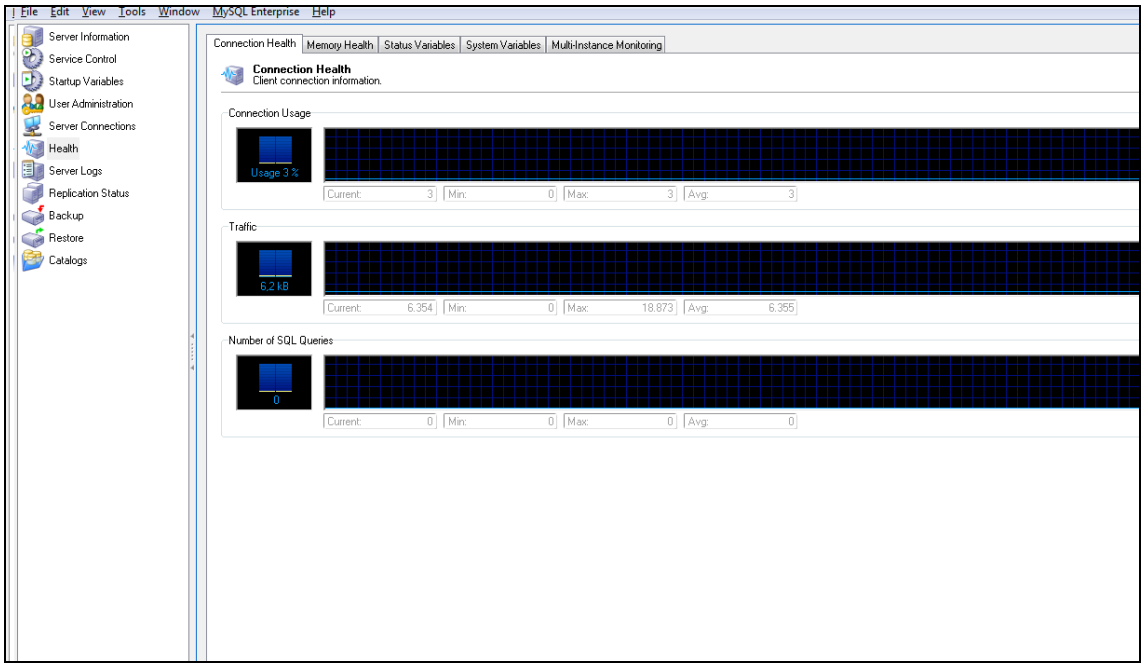


Figura 10-14. MySQL Administrator. Connection Health.

En la imagen que se muestra a continuación se ve el número de transacciones y el tipo de transacción que se ha ejecutado.

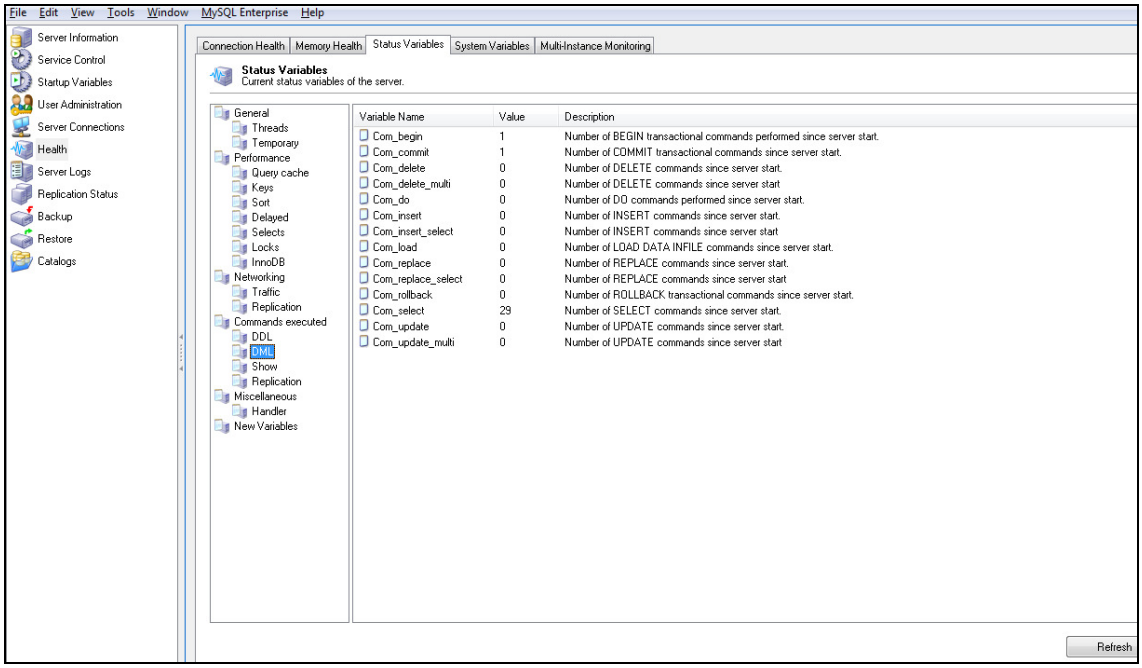


Figura 10-15. MySQL Administrator. Status Variables.

Server Logs

Permite la visualización de *logs*, en caso de estar activos.

Replication Status

Se utiliza en caso de disponer de un sistema de base de datos en modo maestro-esclavo.En este modo cada transacción en la tabla maestra se replica en la tabla esclava. Este tipo de configuración se puede utilizar para hacer balanceo de carga, empleando un tipo de replicación bidireccional, o para tener disponible de forma automática una base de datos idéntica para que las aplicaciones accedan a ella en caso de caída de la principal. En este proyecto no se ha utilizado la funcionalidad de replicación de base de datos de MySQL.

Backup

Permite seleccionar uno o varios esquemas y hacer un *backup* de los mismos. Para ello, la aplicación genera un fichero SQL que contiene los comandos necesarios para crear todas las tablas del esquema y hacer todas las inserciones de las filas que contenga.

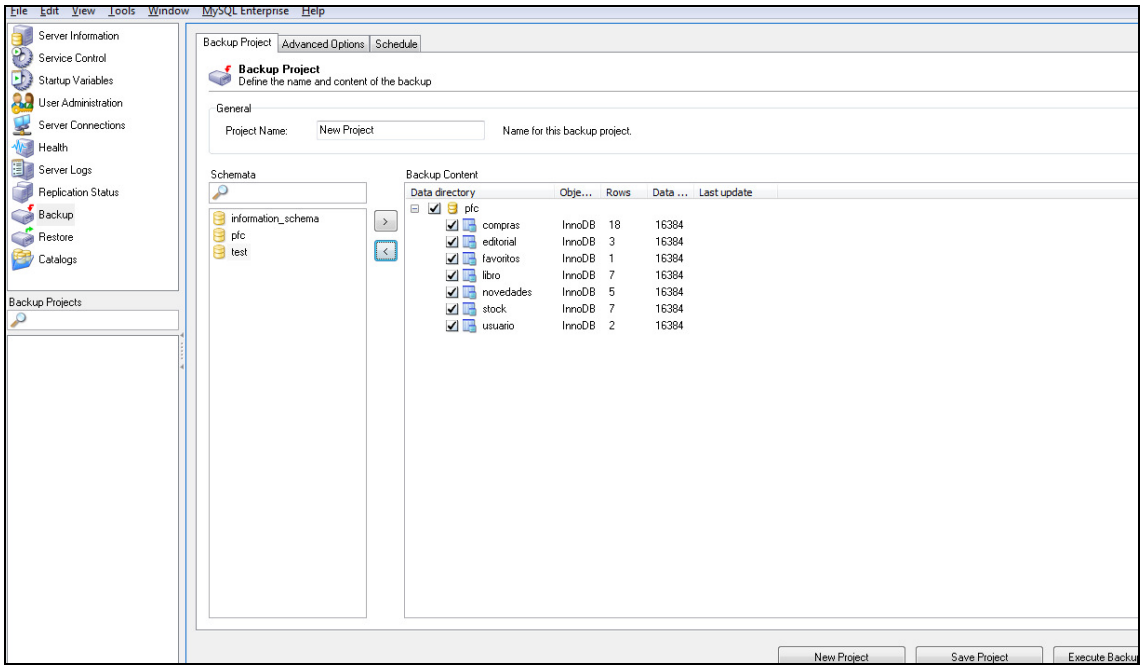


Figura 10-16. MySQL Administrator. Backup.

Restore

Al igual que en la opción de *backup* se genera un fichero SQL con todas las sentencias de creación del esquema, en la opción *restore* la aplicación lee un fichero SQL e interpreta todas las instrucciones para crear el esquema de base de datos.

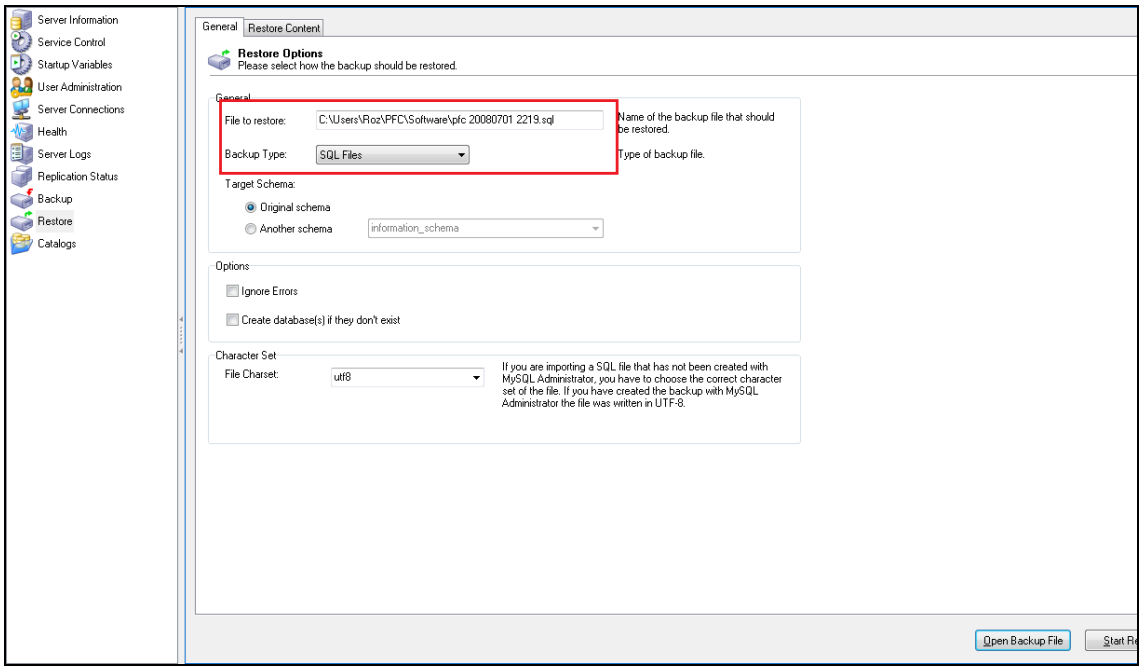


Figura 10-17. MySQL Administrator. Restore.

Catalogs

Desde aquí se crean y gestionan todos los esquemas de base de datos. Se pueden crear y editar tablas, índices, vistas y procedimientos almacenados.

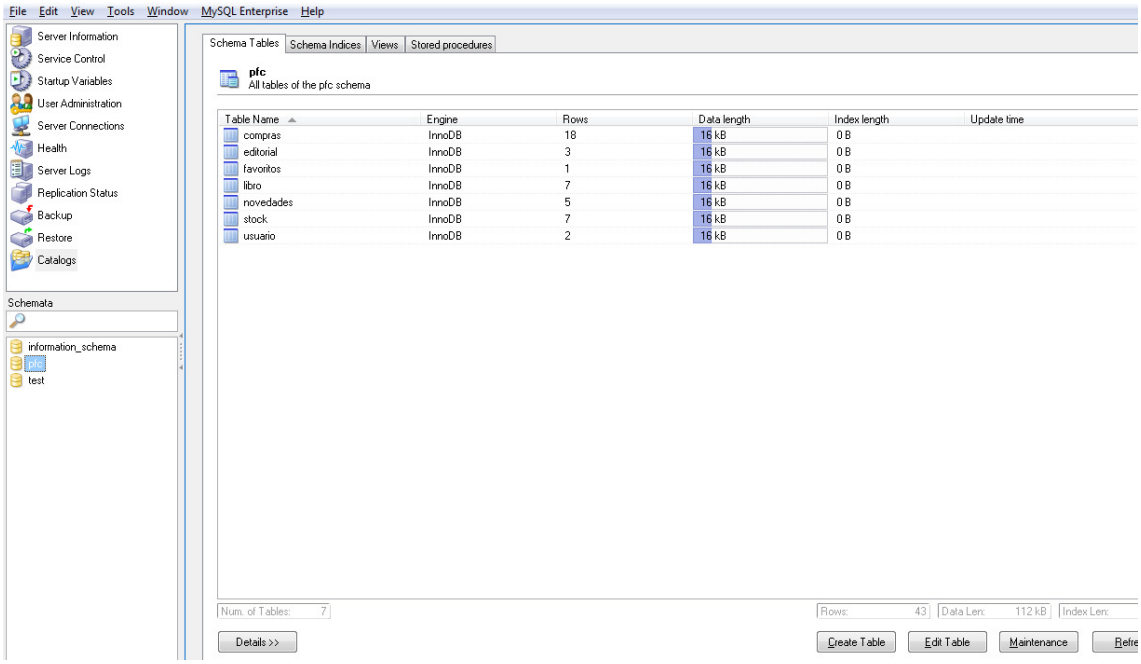


Figura 10-18. MySQL Administrator. Catalogs.

10.4.2. MySQL Query Browser

MySQL Query Browser es una herramienta gráfica proporcionada por MySQL que permite crear, optimizar y ejecutar consultas a la base de datos. Como se ha mostrado, MySQL Administrator está diseñado para administrar el servidor MySQL, y MySQL Query Browser lo está para todo lo relacionado con la consulta y análisis de los datos almacenados en la base de datos.

Aunque todas las consultas se podrían llevar a cabo a través de la línea de órdenes de *mysql*, esta aplicación da al usuario la opción de hacerlo de manera gráfica través de una interfaz más intuitiva.

Diálogo de conexión

Cuando se inicia MySQL Query Browser, aparece el diálogo de conexión, en el que hay que especificar el servidor MySQL al que se desea conectar y las credenciales de acceso. En función de estas credenciales, el usuario tendrá unos permisos sobre las tablas u otros. Además, hay que especificar a qué esquema de la base de datos se quiere acceder. También hay que indicar cuál es el servidor que contiene a la base de datos, en este caso, al estar instalada en la misma máquina desde la que se lanza la aplicación, habrá que indicar *localhost*, pero si está en un

servidor remoto, hay que indicar la dirección IP de dicho servidor. Por defecto, MySQL se instala en el puerto 3306, pero si está funcionando en otro puerto también habría que indicarlo.

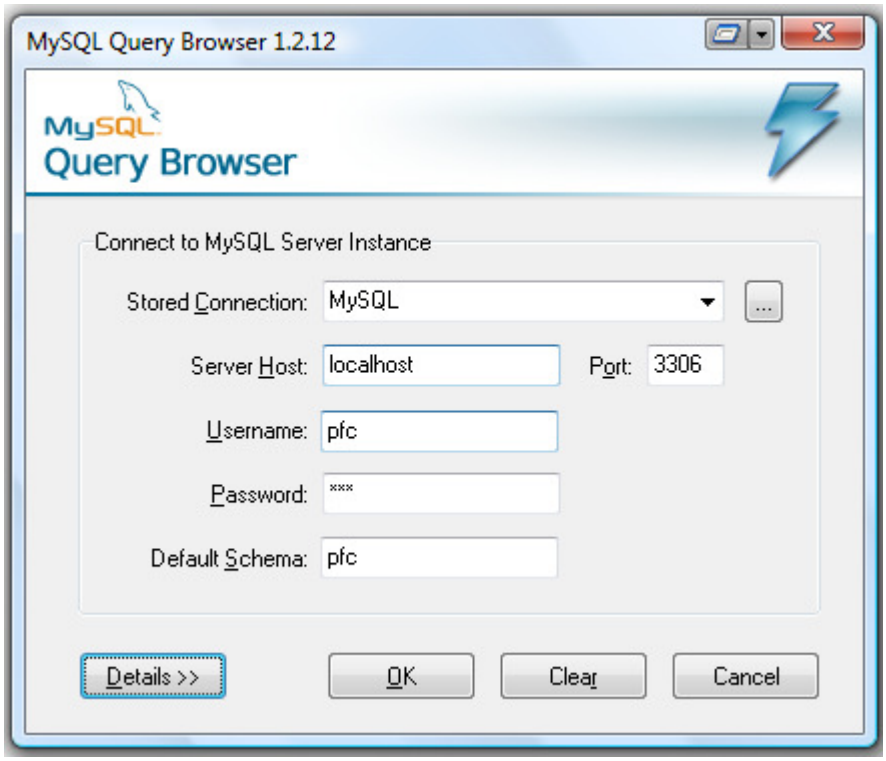


Figura 10-19. MySQL Query Browser. Diálogo de conexión.

Ventana central de consultas

Una vez que se ha podido acceder a MySQL correctamente, se muestra la ventana central de consultas. Desde aquí están diponibles todas las funcionalidades de la aplicación. La siguiente figura es un ejemplo de esta pantalla:

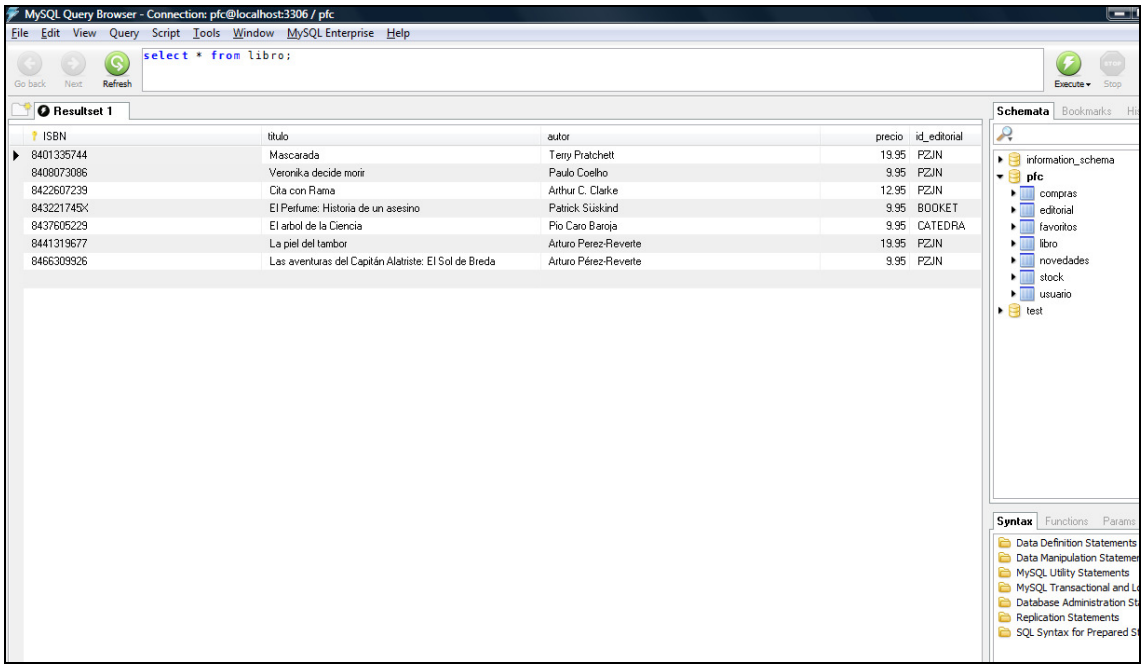
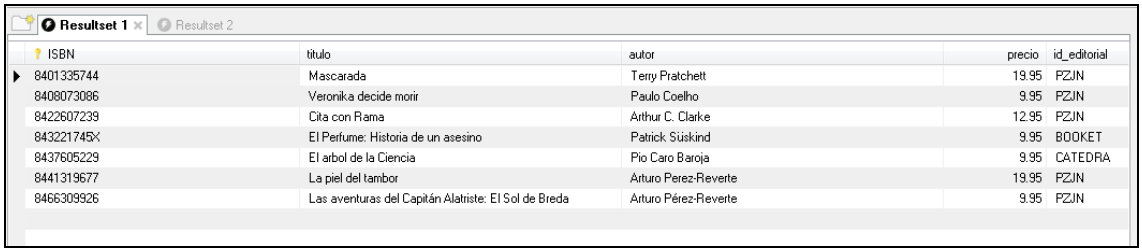


Figura 10-20. MySQL Query Browser. Ventana central de consultas.

En la parte superior se puede ver la barra de herramientas de consulta, donde se crean y ejecutan las sentencias. En la parte de la izquierda hay tres botones de navegación que permiten ir hacia la consulta anterior, hacia adelante y refrescar. En la parte central se escribe la consulta y la parte derecha cuenta con los botones de acción, que sirven para ejecutar o parar la consulta.

En la parte central de la aplicación se puede ver la pestaña de resultado, donde se muestran los resultados de las consultas. Se pueden tener varias pestañas activas a la vez para trabajar con múltiples sentencias.



En la barra lateral se encuentra un navegador de objetos que permite administrar las bases de datos, los favoritos y el historial. Se puede escoger la base de datos y tablas sobre la que operar,

agregar consultas habituales en los favoritos y navegar a través de consultas previamente utilizadas.

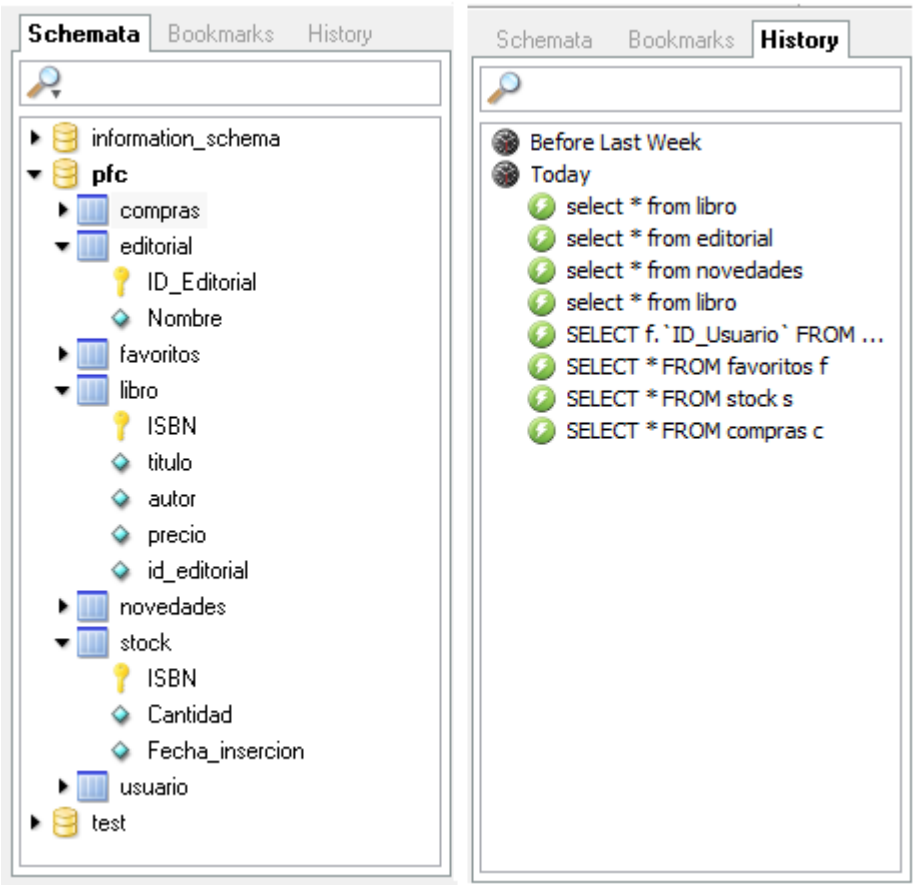


Figura 10-21. MySQL Query Browser. Detalle de la barra lateral.

En la barra lateral hay un navegador de información donde se dispone de funciones más complejas pre-construidas.

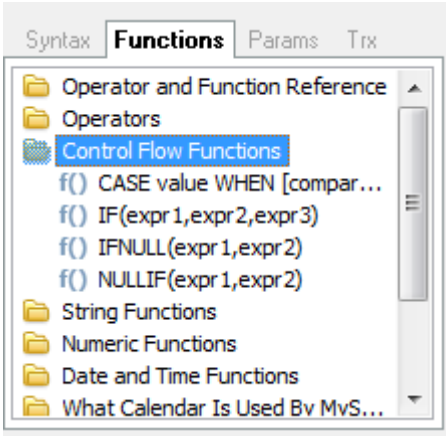


Figura 10-22. MySQL Query Browser. Funciones.

11. Bibliografía

[1] Chuck Cavaness. *Programming Jakarta Struts.* O'Reilly.

[2] James Snell, Doug Tidwell, Pavel Kulchenko. *Programming Web Services with SOAP.* O'Reilly.

[3] Ted N. Husted, Cedric Dumoulin, George Franciscus, David Winterfeldt. *Struts in Action. Building web applications with the leading Java framework.* Manning.

[4] James Holmes. *Struts: The Complete Reference, 2nd Edition.* McGraw-Hill. 2006.

[5] Página oficial de Struts. *http://struts.apache.org.*

[6] Ethan Cerami. *Web Services Essentials.* O'Reilly.

[7] Especificación de SOAP de la W3C. *http://www.w3.org/TR/soap12-part0.*

[8] Robert Englander. *Java and Soap. Building Web Services in Java.* O'Reilly. 2002.

[9] Página oficial de Axis2. *http://ws.apache.org/axis2/.*

[10] Wikipedia: Spring Framework. *http://es.wikipedia.org/wiki/Spring_Framework.*

[11] Craig Walls, Ryan Breidenbach. *Spring in Action.* Manning. 2004.

[12] Chris Schalk, Ed Burns, James Holmes. *Java Server Faces: The Complete Reference.* McGraw-Hill Osborne Media. 2006.

[13] Página official de SUN: Core J2EE Patterns – Data Access Object. *http://java.sun.com/blueprints/corej2eepatterns/Pateterns/DataAccessObject.html*.

[14] Guía sobre CSS de la W3C. *http://www.w3c.es/divulgacion/guiasbreves/ejemplos/CSS/EjemploCSS2.*

[15] Tutorial de Tomcat. *http://www.programacion.net/java/tutorial/tomcatintro*.

[16] Tutorial de Axis2. *http://jcesarperez.blogsome.com/*.

[17] Página oficial de Cocoon. *http://cocoon.apache.org/*.

[18] Sitio oficial de JBoss Seam. *http://jboss.com/products/seam.*

[19] Wikipedia: JBoss Seam. *http://es.wikipedia.org/wiki/JBoss_Seam.*

[20] Wikipedia: Especificación Java Servlets. *http://es.wikipedia.org/wiki/Java_Servlet.*

[21] Deepal Jayasinghe. *Quickstart Apache Axis2: A practical guide to creating quality web services.* Packt Publishing. 2008.

[22] Especificación de Tiles, página official de Struts. *http://struts.apache.org/1.3.8/struts-tiles/index.html.*

[23] Herramienta Axis2 Code Generator Wizard, página official de Axis2. *http://ws.apache.org/axis2/tools/1_0/eclipse/wsdl2java-plugin.html.*

[24] Spring Framework: el contenedor de inversion de control (IoC). *http://static.springframework.org/spring/docs/2.0.x/reference/beans.html*.

[25] Wikipedia: Expression Language (EL). *http://en.wikipedia.org/wiki/Expression_Language*.

[26] Wikipedia: Tomcat. *http://es.wikipedia.org/wiki/Apache_Tomcat.*

[27] Página oficial de Tomcat. *http://tomcat.apache.org/.*

[28] Jason Brittain, Ian F. Darwin. *Tomcat: The definitive Guide.* O'Reilly. 2003.

[29] Sitio Web de Eclipse. *http://www.eclipse.org/.*

[30] Manual de referencia de MySQL. *http://dev.mysql.com/doc/refman/5.0/es/index.html.*

[31] Sitio oficial de MySQL. *http://www.mysql.com.*

[32] Sitio oficial de la comunidad MySQL Hispano. *http://www.mysql-hispano.org/.*

[33] Página oficial de Log4j. *http://logging.apache.org/log4j/1.2/index.html*.

[34] Wikipedia: Log4j. *http://es.wikipedia.org/wiki/Log4j*.

[35] Wikipedia: Algoritmo MD5. *http://es.wikipedia.org/wiki/Algoritmo_MD5*.

[36] Sitio oficial de Turbine. *http://turbine.apache.org/.*